CORE JAVA

SURESH

INeTSOLV

INDEX

Software:

-A general definition of Software is a conversion of a requirement into an application or program.

-A technical definition of Software is it is collection of different programs which designed perform a particular task.

- Program is a set of different instructions
- software will reduce the human task make the business automated
- we have following 2 types of software's

1. System software

- System software If any software is developed for the purpose of making any hardware devices work, such kind of software's are called as system software.
- system software's are generally developed in languages like C, C++, ...

Eg:

os, printer drivers,web cam drivers, Compilers and etc.

2.Applications software

-If any software is developed for the purpose any user to complete his requirements like maintaining his business, for entertainment, for storage called as application software's.

- application software's can be developed by using languages like java,.net , ...

Eg:

notepad, wm player, calc, bank s/w ,hosp s/w ,SM s/w,...

Java is released by sun micro systems guys into the market in following 3 editions

JSE: JSE stands for Java standard edition which can be used for developing stand alone applications.

JEE: JEE stands for java enterprise edition which can be used for developing web based applications.

JME: JME stands for java mobile edition or micro edition which can be used for developing software's for mobile devices or embeded controllers.

In Java we can develop Following 2 types of application software's.

1. Stand alone applications:

- Stand alone applications are available in client system and runs in the same client system which are also called as client side applications
- Stand alone applications are specific to single system which are also called as off-line applications .

Eg:

MS-office, calc,...

2. web based applications:

- web based applications are available in server system and runs in the same client system which are also called as server side applications

-web based applications are not specific to single system which are also called as on-line applications .

- web based application always runs in the context of browser.

Ex: Facebook, Gmail, online games and etc.

Features of java language:

- Every programming language provides some facility's or services which are called as features of that language
- java language provides following various features

1. Simple:

- The java programming language is called as a simple programming languagebecause of the following reasons.

1.The complex topics like pointers, templates, virtual functions,friend functions,.... are eliminated in java language making it simple.

2.The syntax of java language are simple to read , understand and we can easily implement in the application which are similar to syn-taxes of C, C++

2. Architecture Neutral or Platform Independent :

- When an application is developed in java language if it can be executed in any machine without considering the operating system, without considering the Architecture, without considering the software and hardware etc

translators

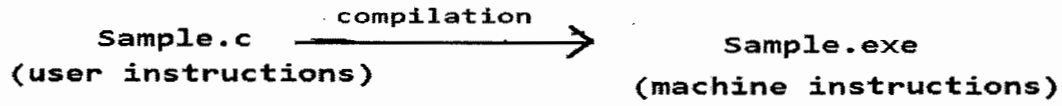
- a translator is a program which is used to translate or convert user instructions of the program into machine understandable instructions so that we can execute our program.
- we have following 3 types of translators

1.Interpreter: Interpreter which will translate the program line by line of program

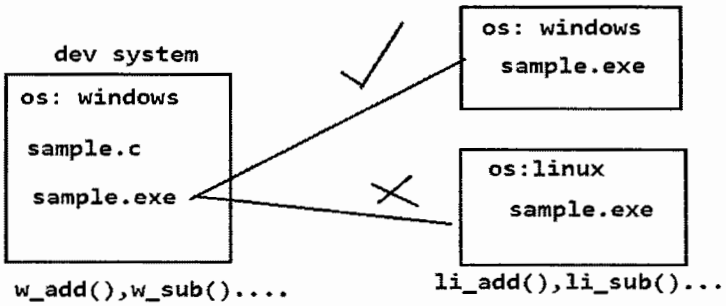
2. compiler: compiler will translate the program all lines the program at a time

3. assembler: assembler will translate assembly language instructions into machine instructions line by line

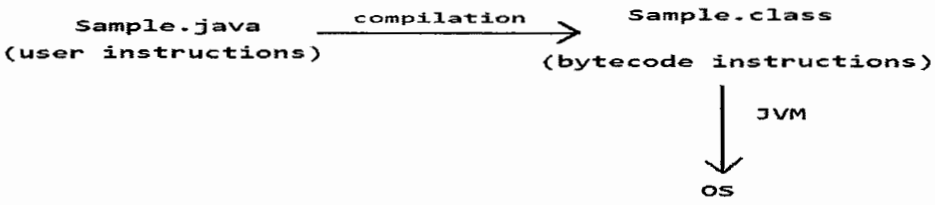
C language does not support this Platform Independent



- When we compile C or C++ programs then compiler will generate .exe files which contains machine instructions
- This compiled code or .exe file can be executed only in compatible machines.
- But If we execute the same .exe file in another incompatible machine it can not be executed hence we can call c or c++ applications are platform or machine dependent.



But Java language support this Platform Independent

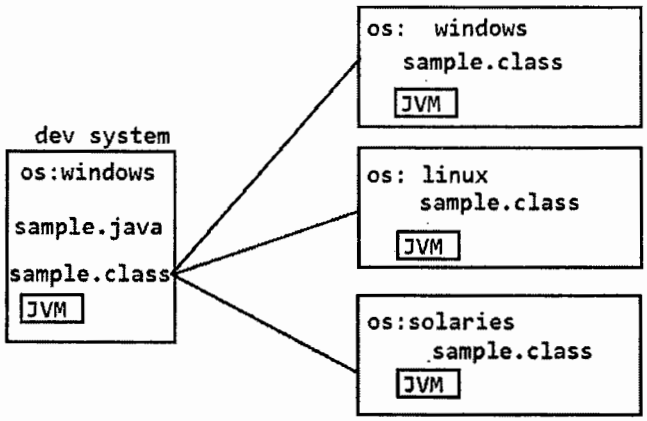


-When we compile java program then java compiler will generate byte code instructions.

-Byte code instructions are not machine instructions

-Byte code instructions are special java instructions which are ready to convert into machine instructions

-we can execute this byte code instructions in both compatible and incompatible machines by taking help of JVM(java virtual machine)hence java applications are called platform independent apps



3. Portable:

- Java language can be called as portable language because using java language can develop reusable components , applications that can be modified ,...

4. Interpreted:

-Java language is called as interpreted language because to execute the java applications in addition to the java compiler we require java interpreter called JVM

5. High performance:

-In earlier versions of java software, java compilation is faster than execution time.

-To improve the performance java people introduced JIT (Just in Time) compilers as a part of JVM.

-It means now java applications are executed by both compiler and interpreter which will reduce the execution time there by providing high performance.

6. Object oriented language:

- Java language is called as an object oriented language because every thing in java is based on a class and an object.
- In every object oriented languages we can contain following 2 components

1.Object

-Anything that exists physically in the real world can be considered as an object.

- Every object will contains some properties and some actions.
- Properties can be used for describing the object by priviing some data.
- Actions represent the task performed by an object.

2.Classes

- A class can be considered as a model or blue print or a plan for creating objects.
- Using a class we can create any number of objects.
- Object creation is not possible without a class.

-A class is a collection of common features of a group of objects.

-A class does not exist physically.

Eg:

Class: Actor

Object: pawan kalyan, rajani kanth

7.Robust:

- Java language is called as Robust or strong programming language because of the following reasons.

1.Memory management:

-In java language there will be no wastage of memory during the allocation time

-all the unused memory will be automatically deallocated by using a special java program called garbage collector so

that, that memory can be allocated to another program there fore we can say that java follows efficient memory management.

2. Exception Handling:

- Sometimes while executing the java program there is a chance of getting run time error or exceptions. Which will terminate the program in middle of the execution it means incomplete execution abnormal termination of program.
- But java language provides huge support by providing keywords and predefined programs to handle th eexceptions.

8. Distributed:

-Java language is called as Distributed language because using java we can develop distributed application which are nothing but web based applications.

-Every application requires information to be retrieved from different machines available in different locations this is possible with the help of distributed apps

- the main advantage of the distributed apps improving the performance of the application by making the data more accessible and more available.

9. Multithread:

-Java language is called as multi threading language because java language we can develop an application by following multi threading approach.

- the main advantage of multi threading is achieving multi tasking which means Executing multiple tasks at the same time simultaneously which will improve the Performance.

10. **Secured:**

Java language is called a secured language because java language by default provides some security programs using which we can make our data secured.

11.**Dynamic**:

Java language is called as dynamic language because in java the memory allocation will be done dynamically as execution time.

structure of java language

In any programming language the program has to be written in a format that is understandable by its compiler, Following is the structure of java language.

syntax:

```
package packagename;
import packagename;
class ClassName{
  variables;
  methods;
  public static void main(String args[ ]){
     statements;
   }
}
```

Rules

1. package statement

- Package statement is used to create user defined packages
- writing package statement is optional
- but when we write a package statement we have to write as a first line in the program
- At most one package statement We can write

2. import statement

-import statement is used to import and use the existing packages both use defined and predefined packages

-writing import statement is optional

-but when we write we must write after package statement and before the class declaration

- using import statement we can import only one package at a time
- If we want to import multiple packages we have to write multiple packages and we can write any number of import statements

3. class Declaration

- class declaration is also optional
- we can write 0 or more number of classes
- class contains a set of variables and methods which are together called as members of the class
- we can write any number of variables and methods inside the class

4. main() method

- java program execution done by JVM and java program execution begin from main() method
- JVM always expect main() method like follows

```
public static void main(String args[]){
      statements;
  }
```

- in main() method we can write any number of statements

procedure to write,save,compile and execute a java program

step1: to write the java program we need to use any editor like notepad, word pad,VI Editor, IDE,...

step2: write the java program according to the requirement

```
class Welcome{
 public static void main(String args[]){
   System.out.println("Welcome to Java World...");
  }
 }
```

step3: save the program with any name but provide extension name as .java(FirstProgram.java)

step4: compiling the java program

- For compiling and executing the java program we have to take the help of cmd prompt
- To compile the java program we have to use java command called " javac "

syntax:

```
javac   programname (with extension)
```

Eg:

```
javac FirstProgram.java
```

- When we compile the java program first it will check whether the code written is valid or not if code is valid then it will generate .class file based on class name.
- In the above program when we compile, compiler will generate " Welcome.class " file

step5: executing the java program

- To execute the java program we have to use a java command called " java "

syntax:

```
java   ClassName (without extension)
```

Eg:

```
java  Welcome
```

o/p:

```
Welcome to Java World
```

Note:

- Before we compile or execute the java program first we have to install a java software.
- Java software is called as open source because it is available at free of cost and we can freely download from internet.
- Before we compile or execute the java program first we have to set path to java installation folder like follows

set path="c:\Program Files\Java\jdk1.6.0_12\bin"

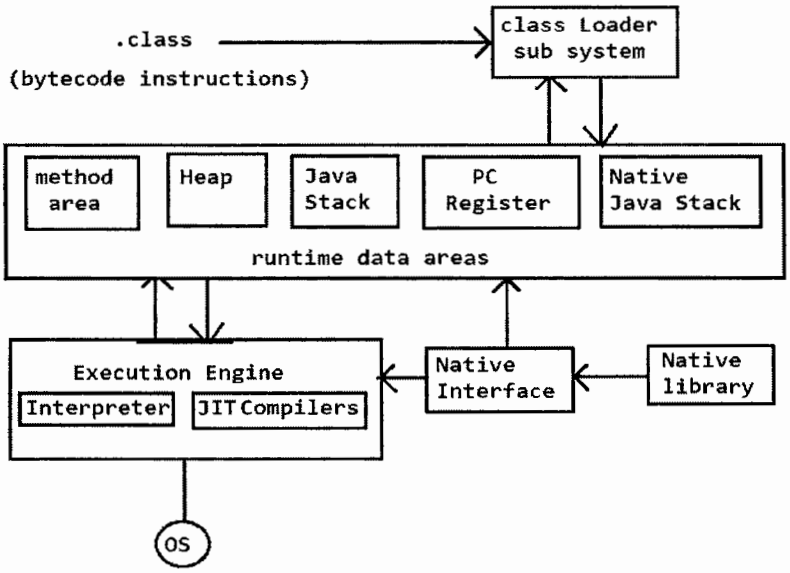
```
//wap to print User Address
class  UserAddress{
public  static void main(String  args[]){
        System.out.println("sachin");
        System.out.println("hno:12-67-56");
        System.out.println("ameerpet");
        System.out.println("delhi");
   }
}
```

Rules

1. we can write any number of statements in a java program which will execute one by one in sequence order.
2. every statement in java should be terminated by " ; "
3. java is called a case sensitive language it means the lower case letters and upper case letters are different.
4. If we don't write main() method then code is valid and compiled successfully but at runtime it will throw a runtime error or exception saying " NoSuchMethodError: main "
5. we can also write an empty java program
6. we can write any number of classes in a single java program and java compiler will generate .class file for every class available in the program.

JVM Architecture

- When we compile the java program, First it will check whether the java program or code is valid or invalid. If program is valid then compiler will generate .class file which contains byte code instructions
- Byte code instructions are the special java instructions which are ready to convert into machine code instructions
- This byte code instructions are executed by JVM (java virtual machine)



1. Class Loader sub system

-Class Loader sub system is the first program available in JVM which is responsible for loading .class file into JVM
-Before loading .class file into JVM it will check whether the .class file is containing valid byte code or not by using a program called " byte code verifier "
- if byte code is valid then it will load .class file into JVM otherwise it wont load .class file

2. Run time data areas

- If byte code is valid then Class Loader sub system will load .class file into following 5 Run time data areas of JVM.

1.Method Area

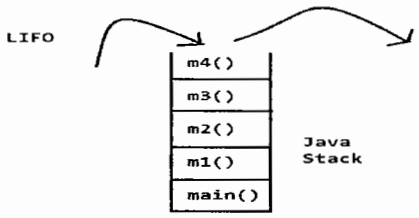
- Method Area contains all Class Code and method Code

2. Heap

-Heap memory area contains all the created objects

3. Java Stack

- Java Stack contains all the methods which are under execution.
- java Stack is a collection of different frames where each frame contains exactly one method.



4. PC(Program Counter) Register

-PC(Program Counter) Register contains address of the next instruction to be executed

-The value of PC(Program Counter) Register will be incremented automatically.

5.Native Java Stack

- Native Java Stack contains all the native methods which are under execution.

3. Native Library

- Native Library is a collection of all predefined non java methods or native methods
- Native method is a method which is written in non java languages like c,c++.

4.Java Native Interface

-Java Native Interface is responsible to load the required Native Methods into Java Native Stack. Sometimes we also use Java Native Interface for loading the required Native Methods into Execution Engine.

5. Execution Engine

- Execution Engine is responsible for converting byte code instructions into machine code instructions.
- Execution Engine contains following 2 parts,

1. Interpreter 2. JIT compilers

-Here both the programs are used at the same time so that performance will be improved

Note:

code is executed by JIT compilers called as " java hotspots "

6.Operating System

Operating system is responsible for allocating the required memory and cpu, other resources

variables

- a variable is a memory location which is used to store values so that we can process the data in the Application

Data Type

-data types are used to create the variables by deciding what type values we are storing, what is the size required and what is the range allowed.

- we have following 3 types of data types,

1. primitive data types

- primitive data types are the basic data types or fundamental data types which are used to store single value.

Eg: int, float,....

2. derived data types

- derived data types are the predefined data types which are used to store multiple values of same type .

Eg: Arrays

3. user defined data types

-user defined data types are defined by user or programmer which are used to store multiple values of different type .

Eg: classes

primitive data types

-Primitive data types are the basic data types or fundamental data types which are used to store single value.

-We have totally 8 primitive data types which are classified into following 4 categories

1. Integer category
2. Floating-point category
3. Character category
4. Boolean Category

1. Integer category

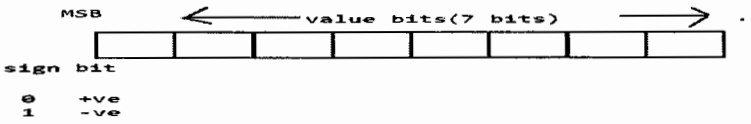
If we write any number either +ve or –ve without any decimal part then it is called as integer.

Eg:

10,-78,23,...

- To store this kind of integers we have to use integer category.
- In this category we have following 4 data types

Data type	size	Range
byte	1(8bits)	-128 to +127 or -2^7 to +2^7-1
short	2(16bits)	-32768 to 32767
int	4(32bits)	-2147483648 to +2147483647
long	8(64bits)	-9223372036854775808 to+9223372036854775807



2. Floating-point category

- If we write any number either +ve or –ve with decimal part then it is called as floating-point values

Eg:

10.7,-78.1,23.9,...

- To store this kind of floating-point values we have to use floating-point category.
- In this category we have following 2 data types

Data type	size	range	No.of Decimals
float	4bytes	1.4E-45 to 3.4E+38	7
double	8bytes	4.9E-324 to 1.8E+308	16

3. Character category

- whenever we write single character between single cotes then it is called as character value

Eg:

'a','A','8','#',...

- To store this kind of character we have to use this character category.
- In this category we have following 1 data type

Datatype	Size	Range
Char	2(16bits)	0-65535

? why java language takes 2 bytes for char datatype where c language takes 1 byte

- **c-language** follows **ASCII** code and which represents English language and to store 0-255 ASCII chars we require 1 byte
- **java – language** follows UNICODE for representing almost all universal languages and to store UNICODE chars we require 2 bytes

4. Boolean Category

- This category is used to store logical values like true or false(in lower case)
- In this category we have following one data type

Datatype	size	range
boolean	JVM dependent	true,false

Note: we never consider 0 or 1 as boolean literals in java these are int literals in java.

Declaration of variables

-Declaration of variables means process of choosing memory locations by choosing type of data,size and required range.

syntax:

datatype var1,var2,var3,....;

Eg:

```
byte age,id;
double salary;
char gender;
boolean check;
```

In java when we declare variables and not initialized with any value then they automatically initialized with default values

Data type	default value
byte	0
short	0
int	0
long	0
float	0.0
double	0.0
char	space
booelan	false

variable initialization

when we declare variables and not initialized with any value then they automatically initialized with default values. But if we want to initialize variables with our own values then we have to go for variable initialization.

Syntax:

datatype var1=value1,var2=value2,var3=value3,....;

Eg:

```
byte age=28,id=34;
double salary=100000.0;
char gender='m';
boolean check=true;
```

variable assignment

-variable assignment means changing the value of the variable

-we always do variable assignment only after declaration of variable

- we can do variable assignment any number of times

Eg:

```
int age=20;
  age=30;
  age=45;
```

literal

A literal is the value that we store into the variable.

keywords

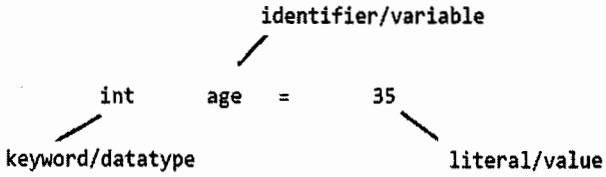
- Keywords are the predefined words or reserved words which are given by java people, which are used to implement different java concepts

-In Java totally 50 keywords which are all defined in lower case

-Each keyword is having its own meaning & purpose that we never change.

identifiers

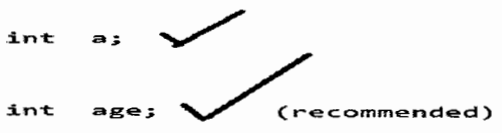
- An identifier is a name given to a class or method or variable, ... To identify and access in the application




```
//wap to demo on variables
class VarDemo{
 public static void main(String args[]){
   int a=10;
  double b=34.56;
   char c='k';
  boolean d=true;
 System.out.println("a value is "+a);
 System.out.println("b value is "+b);
 System.out.println("c value is "+c);
 System.out.println("d value is "+d);
 }
}
```

Rules for identifiers

1. identifier must contain chars only from character set **a-z, A-Z, 0-9, $, _**
2. identifier must not begin with any digit, but it can begin with either alphabet or $ or _ and It must not be any value
3. It must not be any java keyword
4. Length can be anything but it is recommended it should not exceed 16 chars
5. It should be short and meaning full



6. we can also use predefined class names or methods names as identifier but it is not recommended.

Comments

-Comments are used to explain about the program and make the program more readable and understandable

-we can write any number of comments and any where inside the program

-Comments are non executable statements;

- In java we have following 3 types of Comments

1. single line comment

- If we want to write comments in 1 line then we go for this single line comment.

Eg:

```
// ------------------------
```

2. Multi line comment

If we want to write comments in more than 1 line then we go for this multi line comment.

Eg:

```
/* ------------------------
   ------------------------ */
```

3. Document comments

this comments are used to create a separate document which contains information about the program in detail.

Eg:

```
/**------------------------
   ------------------------ */
```

literals

- a literal is a value that is stored into any variable

- In java we have following 6 types of literals

1. int literal

int literal can be created directly by writing any number either positive or negative without decimal part.

Eg:

10,-89,45,.....

- int literal can be stored into any numerical data types directly (byte, short ,int,long,float,double,char)

- we never store int literal into boolean data type either directly or indirectly.

2. long literal

long literal can be created indirectly by suffixing letter l or L along with int literal.

Eg:

10L, 10l, -21L,...

- long literal can be stored into long,float,double directly.

-long literal can not be stored into byte, short ,int,char datatypes directly but we can store indirectly(type casting)

- we never store long literal into boolean data type either directly or indirectly.

Note

1.we never create byte or short literal either directly or indirectly.

2.but if we want to store a value into byte or short datatypes we can use int literal,in this case value must be in therange

double literal

- double literal canbe created in following 2 ways,

1. creating directly by writing the number with decimal part

Eg:

10.2,-10.2,11.1,...

2. creating indirectly by suffixing letter d or D along with any int literal or double literal.

Eg:

10D, -12.1d,12.4D,....

-this double literal can be stored only into double datatype directly

-but if we want to store double literal into other datatypes we can store indirectly (type casting)

float literal

we never create float literals directly but we can create float literals indirectly by suffixing the letter f or F

Eg:

10f, 10.0f,.....

- we can store float iterals into float,double datatypes direclty
- but if we want to store float literal into other datatypes we can store indirectly(type casting)

character literal

-character literals can be created by directly writing any letter in between single quotes.

-character literal can be stored into any numerical dtatypes (byte, short ,int,long,float,double,char)

-character literal can accept any lowercase or upper case alphabet or any digit or any special character.

-character literal can take exactly one character like follows,

Eg:

'a','A','8','@',..

```
char  ch = 'a' ;   ✓
char  ch = 'abc';  ✗
char  ch='';       ✗
char  ch= "a";     ✗
```

character literal can also be created using UNICODE format

syntax:

'\uxxxx' where xxxx can be any 4 digit hexa decimal number (0-9,a-f/A-F)

```
char  ch= '\u0000'   ✓
char  ch= '\U1234'   ✗
char  ch= '\u123'    ✗
char  ch= '1234'     ✗
char  ch= '\uface'   ✓
char  ch= '\u0bad'   ✓
char  ch= '\uABCD'   ✓
char  ch='\uXYZ1'    ✗
```

charcter literal can be created based on escape sequence chars

Escape sequence chars	meaning
'\n'	newline
'\t'	tab space
'\''	'
'\\'	\
'\"'	"
'\u'	Unicode
'\b'	backspace

Eg:

```
System.out.println("D:\\BACKUP\\suresh\\corejava");
```

boolean literal

-In java we have 2 predefined boolean literals that is true or false (in lowercase and these are not keywords)

-boolean literal can be stored only into boolean data type

Note:

we never consider 0 or 1 as boolean literals in java these are int literals in java.

Java coding conventions

In Java For all predefined classes , methods, variables,..... java people already following some coding conventions and recommending us to follow same coding conventions for our own classes , methods, variables,.....

Following are different java coding conventions

1. coding convention for a class

A class name can contain any number of words but every word first letter should be in Capital letter.

Eg:

String, StringBuffer, InputStreamReader, VarDemo,Student,...

2. coding convention for interfaces

An interface name can contain any number of words but everyword first letter should be in capital.

Eg:

Clonable, Runnable, Serializable, ActionListener, MyInterface,.....

3. coding convention for methods

A method name can contain any number of words where first word all letters should be in lowercase and second word onwards every word first letter should be in capital.

Eg:

main(),println(),lastIndexOf(),getAgeOfPerson(),getName(), displayData(),....

4. coding convention for a variables

A variable name can contain any number of words where first word all letters should be in lowercase and second word onwards everyword first letter should be in capital.

Eg:

length,age,endOfTheYear,toDay,.....

5. coding convention for constants

A constant name contains all letters in capital and if we have multiple words in constant name then they should separated by using underscore.

Eg:

PI,MAX_VALUE,MIN_VALUE,EXIT_ON_CLOSE,MY_CONSTANT...

6. coding convention for packages

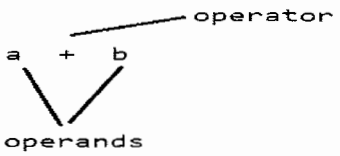
A package name contains all letters in lowercase we may or may not contain multiple words.

Eg:

java,lang,io,awt,mypackage,.....

operators

- An operator is a symbol that is used to perform some operation on any values.
- The data on which we perform operations is called as operands



- In java we perform operations in following 3 levels

1. unary operators

If any operator given with single variable or value or expression then it is called as unary operators.

Eg:

-a, +a,....

2. binary operators

If any operator given with 2 variables or values or expressions then it is called as binary operators.

Eg:

a+b,a-b,a+b+c,.....

3. ternary operators

If any operator given with 3 variables or values or expressions then it is called as ternary operators.

Arithmetical operators

- Arithmetical operators are used to perform different mathematical operations.
- In java we have following 5 types of arithmetical operators

Eg:

+	Addition
-	Subtraction
*	Multiplication
/	Division(Quotient)
%	Division(Remainder)

Rules

1.arithmetical operators can be applied only on numerical data types but we never apply on boolean data types

10+20 ✓

true+true ✗

2.if any mathematical expression is evaluated then the final result datatype can be decided like follows

combination of different expressions	Final result type
byte,byte	int
byte,short,char	int
char,char	int
int,int	int
byte,short,int	int
byte,short,int,long	long
byte,short,int,long,float	float
byte,short,int,long,float,double	double

3. mathematical expression is evaluated based on operator precedence.

Operator precedence (BODMAS)

( )
/ % *
+ -

Eg1:

```
int x=2;
System.out.println(x*10+2); //22
```

Eg2:

```
int x=2;
System.out.println(x*(10+2)); //24
```

Eg3:

```
int x=2;
System.out.println(2-x*10/3); //-4
```

Eg4:

```
int x=2;
System.out.println(x*10/3-2);  //4
```

assignment operators

assignment operators are used to assign or store any value or expression result into left side variable. assignment operators can be specified by using single = symbol

Rules

1. when we write assignment operators then right side we can write anything but left we must write only variable.
2. we can also write assignment operator along with arithmetical operators which are called as compound assignment

```
a=10;
10=a;
a = b+c ;
b+c = a;
a=b;
b=a;
```

Operators, compound assignment operators will do automatic type casting.

```
a=a+b  -->  a+=b
a=a-b  -->  a-=b
a=a*b  -->  a*=b
a=a/b  -->  a/=b
a=a%b  -->  a%=b
```

```
// wap to demo on compound assignment operators
class Sample{
 public static void main(String args[]){
byte a,b;
  a=10;
  b=20;
 // a=(byte)(a+b);  //type casting
  a+=b;  // type casting is done automatically
  System.out.println("Addition="+a);
 }
}
```

- we can also nest the assignment operators

```
int  a,b,c,d;
 a=b=c=d=10;
```

nesting

```
int  a=b=c=d=10;
```

```
int  a=10,b=10,c=10,d=10;
```

not nesting

3. comparison operators or relational operators

- Comparison operators are used to compare any 2 values or to write the condition in the application.
- "condition" always return boolean value either true or false.
- we have following 6 types of comparison operators

<	Less than
>	Greater than
<=	Less than or equals to
>=	Greater than or equals to
==	equals to
!=	not equals to

Rules

1. comparison operators <,>,<=,>= can be applied only on numerical types but we never apply on boolean types
2. comparison operators ==, != (equality operators) can be applied only on boolean types or only on numerical types separately but not on combination of boolean and numerical types

unary minus

unary minus operator is used to change the sign of the given number.

Eg:

```
int a=10;
a = -a;
System.out.println(a); //-10
```

increment or decrement operators

increment or decrement operators are used to increase or decrease the value of the variable by 1.

Eg:

++,--

1. pre increment(++a)

pre increment means first value will be increased and next used in the application.

```
a=10;
p = ++a;
p ->11
a ->11
```

2. post increment(a++)

post increment means first value will be used before increment and next it will be increased.

```
a=10;
p = a++;
p ->10
a ->11
```

3. pre decrement(--a)

pre decrement means first value will be decreased and next used in the application.

```
a=10;
p = --a;
p ->9
a ->9
```

4. post decrement(a--)

post decrement means first value will be used before decrement and next it will be decreased.

```
a=10;
p = a--;
p ->10
a ->9
```

Rules

1. Incr/Decr operators are unary operators always used with single variable.

```
a++;
a++b;
```

2. - we never write Incr/Decr operators along with any value or expression or constant.
   - nesting Incr/Decr operators is not possible

```
7++
(a+b)++
final int a=10;
a++;
int a=10;
++(++a);
```

3. we never apply Incr/Decr operators on boolean data types

```
boolean a=true;
a++;
```

. **logical operators**

- logical operators are used to combine multiple conditions
- logical operators are also used to complement the given condition

Eg:

&,&&,|,||,^,!

logical and operator(single &)

this operator will combine multiple conditions and returns result as true only when all the given conditions are true otherwise it returns false.

syntax:

CON1 & CON2 & CON3 & CON4

Truth table

C1	C2	C1&C2
T	F	F
F	T	F
T	T	T
F	F	F

here single & operator will evaluate all the conditions and finally returns the result either true or false.

Eg:

```
int a=10;
int b=2;
System.out.println(a<b&++a>++b); //false
System.out.println("a="+a); //11
System.out.println("a="+b); //3
```

logical and operator(double &&)

syntax:

```
CON1 && CON2 && CON3 && CON4
```

Truth table

C1	C2	C1&&C2
T	F	F
F	T	F
T	T	T
F	F	F

here double && operator is called as short circuit operator which will not evaluate all the conditions at a time, it will evaluate first condition if it is false then it returns false directly otherwise if first condition is true then it will evaluate the second condition if it is true then third condition,... and finally returns the result. so that the performance of the application will be improved.

Eg:

```
int a=10;
int b=2;
System.out.println(a<b&&++a>++b); //false
System.out.println("a="+a); //10
System.out.println("a="+b); //2
```

logical or operator(single pipe symbol |)

this operator will combine multiple conditions and returns result as true when any of the given conditions is true otherwise it returns false.

syntax:

```
CON1 | CON2 | CON3 | CON4
```

Truth table

C1	C2	C1|C2
T	F	T
F	T	T
T	T	T
F	F	F

here this | operator will evaluate all the conditions and finally returns the result either true or false.

logical or operator (double pipe symbol(||))

syntax:

```
CON1 || CON2 || CON3 || CON4
```

Truth table

C1	C2	C1||C2
T	F	T
F	T	T
T	T	T
F	F	F

here double || operator is called as short circuit operator which will not evaluate all the conditions at a time, it will evaluate first condition if it is true then it returns true directly otherwise if first condition is false then it will evaluate the second condition if it is false third condition,... and finally returns the result as either true or false. so that the performance of the application will be improved.

logical exclusive-or (xor) operator(^)

this operator will combine multiple conditions and returns result as false when all the given conditions are same otherwise it returns true.

syntax:

CON1 ^ CON2 ^ CON3 ^ CON4

Truth table

C1	C2	C1^C2
T	F	T
F	T	T
T	T	F
F	F	F

here xor(^) operator will evaluate all the conditions and finally returns the result either true or false.

logical not operator(!)

this operator will complement the given condition or expression.

syntax:

!(condition or combinations of conditions)

Truth table

C	!C
T	F
F	T

Bitwise operators

- Bit wise operators are used to perform operations the bits of the given decimal numbers
- we have following bit wise operators in java

&, |, ^, <<, >>, >>>, ~

```
  bitwise & operator
-----------------------
  This operator used to perform bitwise & operation between the bits
of the given 2 decimal numbers
  Eg:
       int a=10;
       int b=3;
       System.out.println(a&b); //2
```

truth table

b1	b2	b1&b2
0	1	0
1	0	0
0	0	0
1	1	1

```
  a=10 -    0    0    0    0    1    0    1    0
  b=3 -     0    0    0    0    0    0    1    1
  ---------------------------------------------------
  a&b -     0    0    0    0    0    0    1    0     => 2
```

bitwise | operator:

This operator used to perform bitwise operation between the bits of the given 2 decimal numbers

Eg:
```
int a=10;
int b=3;
System.out.println(a|b); //11
```

truth table

b1	b2	b1|b2
0	1	1
1	0	1
0	0	0
1	1	1

```
a=10 -    0    0    0    0    1    0    1    0
b=3 -     0    0    0    0    0    0    1    1
-------------------------------------------------  ------
a|b -     0    0    0    0    1    0    1    1 => 11
```

bitwise ^ operator:

This operator used to perform bitwise operation between the bits of the given 2 decimal numbers

Eg:
```
int a=10;
int b=3;
System.out.println(a^b);//9
```

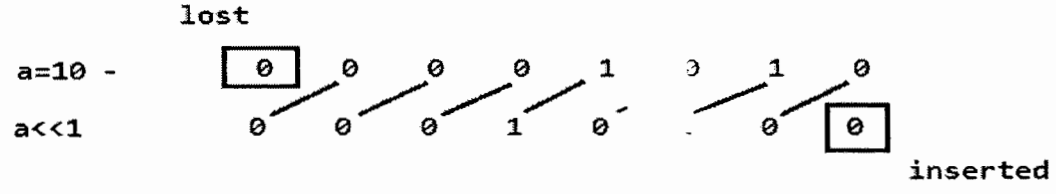
truth table

b1	b2	b1^b2
0	1	1
1	0	1
0	0	0
1	1	0

```
a=10 -    0    0    0    0    1    0    1    0
b=3 -     0    0    0    0    0    0    1    1
-------------------------------------------------  -----
a^b -     0    0    0    0    1    0    0    1   => 9
```

bitwise left shift(<<) operator
---
this operator shift specified number of bits of the given decimal number towards leftside.

Eg:
```
int a=10;
System.out.println(a<<1);//20
```



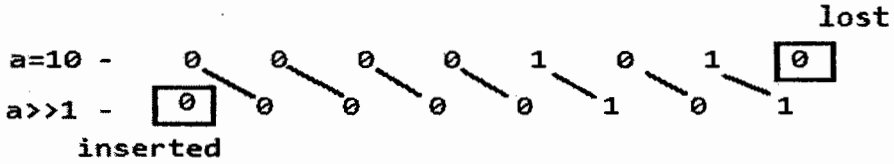
here shifting the bits one by one of the decimal number towards left side is equalto multiplying the number by 2^1.

bitwise right shift(>>) operator

this operator shift specified number of bits of the given decimal number towards right side.

Eg:

```
int a=10;
System.out.println(a>>1); //5
```



here shifting the bits one by one of the decimal number towards right side is equalto dividing the number by 2^1.

unsigned right shift(>>>) operator

this operator works same like as right shift operator only but it allways return positive value.

bitwise ~ operator (negation)

this operator is used to complement the bits of the given decimal numbers it means we change all 0's into 1's and vice versa.

Eg:

```
int a=10;
System.out.println(~a);//-11
```

```
a=10 -  0  0  0  0  1  0  1  0
 ~a  -  1  1  1  1  0  1  0  1
```

but here the result containing msb as 1 which means it is a negative number but a negative number always represented in 2's complement

2's Complement = 1's complement + 1

```
Result is ->  1  1  1  1  0  1  0  1
1's complement -> 0  0  0  0  1  0  1  0
add 1     ->                       1
          ----------------------------------------
             0  0  0  0  1  0  1  1  =  -11
```

or | ~x = -(x+1) |

ternary oeprator or ?: operator or conditional operator

this operator returns the value based on given condition.

syntax:

(condition)?value1:value2;

here if condition is true then it returns value1 otherwise if condition is false then it returns value2.

Eg1:

```
int a=10;
int b=2;
int c=(a>b)?a:b;
System.out.println(c+"Is Big Number");
```

Eg2:

```
int a=10;
int b=20;
int c;
c = (a>b)?a-b:b-a;
System.out.println("Subtraction="+c);
```

- we can also nest the ternary operator

Eg3:

```
int a=10;
int b=2;
int c=6;
int d;
d=(a>=b&&a>=c)?a:((b>=c&&b>=a)?b:c);
System.out.println(d+"Is Big Number");
```

new operator

-new is java keyword or operator which is used to create the object

-we can create an object for both user defined classes and predefine classes

-creating an object nothing but allocating the memory so that we can use in the application.

-once an object created that will be located inside the Heap memory of JVM.

syntax:

ClassName referencevariable = new ClassName();

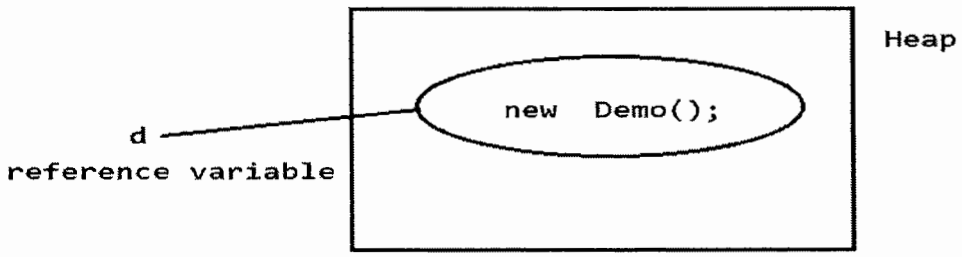
Eg:

Declaration of class

```
class Demo{
        }
```

Declaration of object

```
Demo d = new Demo();
```



. (Dot) operator

- This operator used to access the members of the class using reference or column like follows

Eg:

```
reference.variable
reference.method()
ClassName.variable
ClassName.method()
```

- this operator can also be used to refer or identify the class from a package

Eg:

```
java.lang.NoSuchMethodError
java.lang.String
.....
```

control statements (flow control)

- In java we can write any number of statements which are executed in sequence order By default.
- But if we want to execute java statements randomly according to our requirement then we have to use control statements
- In java we have following 3 types of control statements
  1. conditional statements
     1. if..else
     2. switch-case
  2. iterative statements
     1. for
     2. while
     3. do..while
     4. foreach(not a java keyword)
  3. transfering statements
     1.break
     2.continue
     3.return

1. conditional statements

conditional statements are the control statements which are used to execute group of statements based on condition or value in java we have following 2 types of conditional statements

1.if..else (based on condition)
2.switch-case (based on value)

1. if .. else

this conditional statement used to execute the group of statements based on given condition.

syntax:

```
if(condition){
   statements1;
  }
else{
   statements2;
  }
```

here if condition is true then it will execute statements1 otherwise if condition is false then it will execute statements2

//wap to demo on if..else

```
class IfDemo1{
```

```
public static void main(String args[]){
        int a,b;
        a=100;
        b=100;
        if(a>b){
        System.out.println("a is big");
        }
        else{
        System.out.println("b is big");
        }
  }
}
```

Rules

1. we can execute either if block or else block but we never execute both the blocks at a time
2. we can skip either if block or else block but we never skip both the blocks at a time.
3. For if statement writing else block is optional but we can not write else block directly without if statement.
4. In If statement writing condition or expression mandatory which must be boolean type

```
int a=10;
int b=2;
if(a>b){
}
```

```
boolean b=true;
if(b){
}
```

```
if(){
}
```

```
if(true){
}
```

```
boolean x=true;
   if(x=true){
   }
```

```
int a=10;
if(a){
}
```

```
int a=10;
if(a=10){
}
```

```
boolean x=true;
   if(x==true){
   }
```

```
 if(1){
 }
 else{
 }
```

```
int  a=10;
if(a==10){
}
```

5. If we write more than one statement writing boundaries({,}) are required otherwise if we write 1 statement then writing boundaries({,}) are optional.
6. If we don't write boundaries({,}) then,
   -> we can write only one statement
   -> writing this single statement is mandatory
   -> this single statement must not be any declarative statement

```
int a=10;
int b=2;
if(a>b){
}
```

```
int a=10;
int b=2;
if(a>b)
int x=10;
```

```
int a=10;
int b=2;
if(a>b)
System.out.println("Addition="+(a+b));
```

```
int a=10;
int b=2;
if(a>b) {
int x=10;
}
```

```
int a=10;
int b=2;
if(a>b)
```

```
int a=10;
int b=2;
if(a>b)
 a=100;
```

```
// wap  tocheck whether the given number is even or odd
class  IfDemo2{
 public  static  void  main(String args[]){
          int a =7;
          if(a%2==0){
          System.out.println(a+" is  a Even Number ");
          }
          else{
          System.out.println(a+"is a Odd Number ");
        }
  }
}
//wap to check whether the given number is Positive or negative
class  IfDemo3{
 public  static  void  main(String args[]){
          int a =0;
          if(a>0){
          System.out.println(a+" is  a Positive Number ");
          }
          else
          if(a<0){
          System.out.println(a+" is  a Negative Number ");
          }
          else{
          System.out.println( "it is  just zero ");
          }
    }
}
//wap to read the values from keyboard
import  java.io.*;
class  ReadingData{
 public  static  void  main(String args[]) throws IOException{
      BufferedReader br  =new   BufferedReader( new InputStreamReader(System.in));
           System.out.println("Enter  Any string");
           String  str =   br.readLine();
           System.out.println("str="+str);
           System.out.println("Enter  Any Integer");
           int  i = Integer.parseInt(br.readLine()); //parsing
           System.out.println("i="+i);
           System.out.println("Enter  Any Double");
           double d = Double.parseDouble(br.readLine());
           System.out.println("d="+d);
           System.out.println("Enter  Any character");
           char  ch = (char) br.read(); //type casting
           System.out.println("ch="+ch);
  }
 }
```

```
//wap to find big number in given 3  numbers
import  java.io.*;
class  IfDemo4{
public  static  void  main(String  args[]) throws IOException{
        BufferedReader  br  = new BufferedReader(new InputStreamReader(System.in));
         System.out.println("Enter Any 3 Numbers");
         int a =  Integer.parseInt( br.readLine());
         int b =  Integer.parseInt( br.readLine());
         int c =  Integer.parseInt( br.readLine());
        if(a>=b&&a>=c){
         System.out.println(a+" Is Big Number");
        }
        else
        if(b>=c&&b>=a){
         System.out.println(b+" Is Big Number");
        }
        else{
         System.out.println(c+" Is Big Number");
        }
   }
}
//wap to check whether the given character is vowel or not
import  java.io.*;
class  IfDemo5{
public  static  void  main(String  args[]) throws IOException{
        BufferedReader  br  = new BufferedReader(new InputStreamReader(System.in));
         System.out.println("Enter Any Character");
        char ch  = (char) br.read();
        if(ch=='a' || ch=='e' || ch=='i' || ch=='o' || ch=='u'){
         System.out.println(" It Is Vowel");
        }
        else{
        System.out.println(" It Is Not Vowel");
        }
   }
}
```

assignments

? wap to check whether the user is eligible for voting or not

? wap to check whether the given number is multiple of 5 or not

?wap to check whether the given character is alphabet or not

? wap to find bill amount according to the following information

Amount	Discount
<5000	10%
>=5000 and <10000	15%
>=10000 and <20000	20%
>=20000	50%

take qty, rate and calculate amt, dis and finalbill

2. switch-case

switch-case is another conditional statement which is used to execute the statements based on value.

syntax:

```
switch(variable){
 case label1: statements1;
        break;
 case label2: statements2;
        break;
 .....
 default: statements;
 }
```

here if the value of the variable is equals to any one of the case labels then the corresponding case statements will be executed, otherwise if all the cases are failed then it will execute default statements.

//wap to demo on switch-case

```
class Switch1{
    public static void main(String args[]){
     int choice=4;
     switch(choice){
      case 1: System.out.println("I Like Black Colour");
              break;
      case 2: System.out.println("I Like Merun Colour");
              break;
      case 3: System.out.println("I Like Blue Colour");
              break;
      case 4: System.out.println("I LikeYellow Colour");
              break;
      default: System.out.println("I Dont Like Any Colour");
      }
  }
}
```

Note:

-We can say switch-case is replacement to the if..else..if ladder when it contains many equal comparisons

-But if..else..if ladder contains any other >,<,!=,>=,<= comparisons then we can not say switch-case is the replacement to if..else..if ladder.

-Generally we go for switch-case when we want to make multiple equal comparisons

//write the above program using if..else..if ladder

Rules:

1. Writing the expression or value for switch-case is mandatory which must be any byte/ short/ int/ char, but it should not be any long/ float/ double/ boolean

```
switch(){
 }
```

```
byte a=10;
siwtch(a){
}
```

```
int a=10;
switch(a){
}
```

```
int a=1;
switch(a>1){
}
```

```
long a=1;
switch(a){
}
```

```
int a=1;
switch(a==1){
}
```

```
boolean a=true;
 switch(a){
 }
```

```
int a=1;
switch(a=1){
}
```

2.In switch -case we write any number of statements every time we have to provide boundaries({,})it means mandatory

```
int a=10;
switch(a){
}
```

```
int a=10;
switch(a){
case 1:System.out.println("hi");
}
```

```
int a=10;
switch(a)
```

```
int a=10;
switch(a)
case 1:System.out.println("hi");
```

3. In switch-case writing case statements or default statements are optional. But if we want to write any statement we have to write inside the case statements or default statements.

```
int a=1;
switch(a){
   }
```

```
int a=1;
switch(a){
System.out.println("hi");
  }
```

```
int a=1;
switch(a){
case 1: System.out.println("hi");
  }
```

4.-In switch-case we can write default statement any where inside the switch-case but it is recommended to write default statements in the ending section of the switch-case.

- At most 1 default statement we can write

```
int a=1;
switch(a){
case 1:
case 2:
default:
case 3:
case 4:
}
```

```
int a=1;
switch(a){
default:
case 1:
case 2:
default:
case 3:
case 4:
}
```

//CTE: duplicate default label

```
int a=1;
switch(a){
case 1:
case 2:
case 3:
case 4:
}
```

5. -In switch-case we can write any number of case statements if write a case its label must be specified which should not be duplicated

- In a case we can write any number of statements

```
int   a=1;
switch(a){
case 1:
case 2:
case 3:
}
```

```
int a=1;
switch(a){
case 2:
case 3:
case :
}
```

```
int a=1;
switch(a){
case 1:
case 'a':
case 2:
case 'z':
}
```

```
int   a=1;
switch(a){
case 1:
case 2:
case 3:
case 2:
case 4:
}
```

```
int a=1;
switch(a){
case 65:
case 1:
case 2:
case 'A':
case 4:
}
```

```
byte a=1;
switch(a){
case 1:
case 2:
case 300:
}
```

6. By default switch-

case will enter into any case if case label is successfully compared and execute until end of the switch-case or until the break statement found this nature of switch-case is called fall-through behavior

- But if we want to stop this continuity of switch-case and execute the statements according to our requirement we have to use break statement
- break is transferring statement which will stop the continuity of switch-case and transfer the control from inside the switch-case to outside the switch case

```
//wap to perform all arithmetical operations using switch-case
import java.io.*;
class Switch2{
  public static void main(String args[]) throws IOException{
        BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
        System.out.println("Enterr Any 2 Numbers");
         int a= Integer.parseInt(br.readLine());
          int b=Integer.parseInt(br.readLine());
        System.out.println("1.Addition");
        System.out.println("2.Subtraction");
        System.out.println("3.Multiplication");
        System.out.println("4.Quotient");
        System.out.println("5.Remainder");
        System.out.println("Enter Your Choice");
         int choice=Integer.parseInt(br.readLine());
         switch(choice){
          case 1: System.out.println("Addition="+(a+b));
                  break;
          case 2: System.out.println("Subtraction="+(a-b));
                  break;
          case 3: System.out.println("Multiplication="+(a*b));
                  break;
          case 4: System.out.println("Quotient="+(a/b));
                  break;
          case 5: System.out.println("Remainder="+(a%b));
                  break;
          default: System.out.println("Invalid choice");
          }
    }
}
// wap to find whether the given character is vowel or not using switch-case
import java.io.*;
class Switch3{
  public static void main(String args[]) throws IOException{
        BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
        System.out.println("Enter Any Character");
       char ch= (char) br.read();
         switch(ch){
          case 'a':
          case 'e':
          case 'i':
          case 'o':
          case 'u':
```

```
        case 'A':
        case 'E':
        case 'I':
        case 'O':
        case 'U': System.out.println("It is vowel");
                  break;
        default: System.out.println("It is not a vowel");
        }
    }
}
```

assignments

- Wap to perform all arithmetical operations using switch-case by taking arithmetical operator.
- Wap to display month name for given month number
- Wap to display the season of the given month
- Wap to calculate the Bill according to following information

Room Type	Per Day charge
Suit	1800 rs/-
Deluxe	1350 rs/-
Ac	1200 rs/-
Ordinary	800 rs/-

2. iterative statements or loops

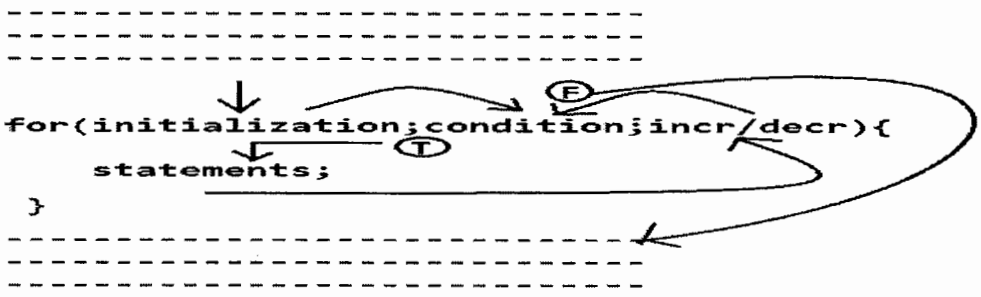
-iterative statements are the control statements which are used execute group of statements for n number of times.

- in java we have following 4 types of loops
  1. for
  2. while
  3. do..while
  4. foreach (not a java keyword) (java1.5)

1. for

this loop is recommended to use when we know the number of iterations exactly.

Syntax:



In the above syntax first control will execute the statements given before the for loop next it will enter into first section of the for loop that is initialization section, next it will goto condition section if condition is true then it will enter into for body execute statements after the completion of execution it will enter into third section of the for loop that is Incr/Decr section, next it will again enter into condition section if condition is true again it will execute the statement this procedure will be continued until the condition becomes false.

```
//wap to demo on for loop and print 1-10 numbers
class For1{
 public static void main(String args[]){
  for(int i=1;i<=10;i++){
    System.out.println(i);
```

```
  }
 }
}
//wap to print Even Numbers between 1-100
class For2{
 public static void main(String args[]){
 for(int i=1;i<=100;i++){
   if(i%2==0)
    System.out.println(i);
 }
 }
}
```

Rules

1. In for loop 3 sections must be separated by using 2 " ; " symbols
2. writing 3 sections in for loop are optional.
   - if we dont write anything in initialization section then compiler wont write anything.
   - if we dont write anything in Incr/Decr section then compiler wont write anything.
   -But if we dont write anything in condition section then compiler always write boolean value true.

```
for(,,){ ✗        for(;;){ ✓ infinite loop
  }                 }
```

3.-In initialization section we can write any valid java statements
   -In Incr/Decr section also we can write any valid java statements wont write anything.
   -But In condition section we can write anything which must be boolean type

Eg:

```
class ForTest{
  public static void main(String args[]){
int i=1;
for( System.out.println("HI");i<=10; System.out.println("java")){
 System.out.println(i);
 i++;
 }
 }
}
```

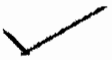
4.-In initialization section we can write any number of initializations which must be same type
 -In Incr/Decr section also we can write any number of incr/decr statements which must be separated by ","
 -In condition section also we can write any number of conditions which must be combined using logical operators

```
for(int i=1,double j=10;i<=10;i++,j--){ ✗
  System.out.println(j);
 }

for(int i=1,j=10;i<=10;i++ j--){ ✗
  System.out.println(j);
 }

for(int i=1,j=10;i<=10,j>5;i++,j--){ ✗
  System.out.println(j);
 }

for(int i=1,j=10;i<=10 && j>=5;i++,j--){ ✓
 System.out.println(j);
 }
```

5. variables declared inside the for loop we can not access out side the for loop

```
for(int i=1;i<=10;i++){
System.out.println(i);
}
System.out.println(i);
```

```
int i=1;
for(;i<=10;i++){
System.out.println(i);
}
System.out.println(i); //11
```



6. In for loop if we write multiple statements then writing curly braces are mandatory otherwise if we write single statement then writing curly braces are optional, but in this case writing statement is mandatory and it should not be any declarative statement.

```
//wap to find factorial of the given number
import java.io.*;
class For3{
 public static void main(String args[]) throws IOException{
BufferedReader kbd = new BufferedReader(new InputStreamReader(System.in));
System.out.println("Enter Any Number");
 int n=Integer.parseInt(kbd.readLine());
 int f=1;
 for(int i=1;i<=n;i++){
    f = f * i;
   }
  System.out.println(n+" != "+f);
  }
}
//wap to print mathematical table of given number
import java.io.*;
class For4{
 public static void main(String args[]) throws IOException{
BufferedReader kbd = new BufferedReader(new InputStreamReader(System.in));
System.out.println("Enter Any Number");
 int n=Integer.parseInt(kbd.readLine());
 System.out.println("Mathematical of "+n);
 for(int i=1;i<=10;i++){
   System.out.println(n+"X"+i+"="+n*i);
  }
  }
}
//wap to check whether the given number is prime number or not
import java.io.*;
class PrimeNumber{
 public static void main(String args[]) throws IOException{
BufferedReader kbd = new BufferedReader(new InputStreamReader(System.in));
System.out.println("Enter Any Number");
 int n=Integer.parseInt(kbd.readLine());
  int c=0;
 System.out.println("Factors of "+n);
 for(int i=1;i<=n;i++){
 if(n%i==0){
   System.out.println(i);
   c++;
```

```
   }
  }
  if(c==2){
     System.out.println("It is a PrimeNumber");
   }
  else{
     System.out.println("It is Not a  PrimeNumber");
 }
}
}
```

assignments

*Wap to calculate sum of 1-100 numbers
*Wap to print 1-100 Even numbers without using % operator
*Wap to print 1-100 in reverse order
*Wap to print power of given 2 numbers

while loop

while loop is recommended when we dont know how many times to repeat.

syntax:

```
------------------
------------------
initialization;
while(condition){
  statements;
  incr/decr;
 }
------------------
------------------
```

here first control will execute tatements given before the while loop and next it will come to condition section of the while loop and check for the result of the condition if condition is true then it will enter into the loop and execute all the statements after the completion of execution again it will go to condition if it is true again it will execute the statements this procedure will be continued until the condition becomes false.

```
//wap to print 1-10 numbers using while loop
class  While1{
 public  static  void  main(String args[]){
    int i=1;
 while(i<=10){
  System.out.println(i);
  i++;
  }
 }
}
```

Rules

1.In while loop writing the condition or expression is mandatory which must be boolean type

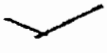
```
while(){
 }
```

```
boolean i=true;
while(i){
 }
```

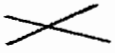


```
int i=1;
while(i){
 }
```

```
while(true){
 }
```



```
while(1){
 }
```



2. In while loop if we write multiple statements then writing curly braces are mandatory otherwise if we write single statement then writing curly braces are optional, but in this case writing statement is mandatory and it should not be any declarative statement.

3.In java every statement should get a control and execute at any 1 point of time and compiler never support unreachable statements

```
//wap to find whether the given number is perfect number or not
import java.io.*;
class While2{
 public static void main(String args[]) throws IOException{
BufferedReader kbd = new BufferedReader(new InputStreamReader(System.in));
System.out.println("Enter Any Number");
 int n=Integer.parseInt(kbd.readLine());
 int sum=0;
 System.out.println("Factors of "+n);
 int i=1;
 while(i<=n/2){
   if(n%i==0){
    System.out.println(i);
   sum=sum+i;
   }
  i++;
  }
  if(sum==n){
     System.out.println("It is a Perfect Number");
  }
 else{
  System.out.println("It is Not a Perfect Number");
  }
 }
}
//wap to find sum of the digits of given number
import java.io.*;
class While3{
 public static void main(String args[]) throws IOException{
BufferedReader kbd = new BufferedReader(new InputStreamReader(System.in));
System.out.println("Enter Any Number");
 int n=Integer.parseInt(kbd.readLine());
int sum=0;
 int r;
  while(n!=0){
  r=n%10;
   sum=sum+r;
  n=n/10;
   }
System.out.println("sum of the digits: "+sum);
  }
}
```

assignments

//wap to find big digit in the given number

```
// Wap to print power of given 2 numbers
import java.io.*;
class DoWhile2{
 public static void main(String args[]) throws IOException{
BufferedReader kbd = new BufferedReader(new InputStreamReader(System.in));
System.out.println("Enter N Number");
 int n=Integer.parseInt(kbd.readLine());
System.out.println("Enter Power value");
 int p=Integer.parseInt(kbd.readLine());
 int i=1;
 int res=1;
  do{
 res = res * n;
    i++;
   }while(i<=p);
 System.out.println(n+ "to the power of"+p+" is "+res);
}
}
//wap to find the multiplication of 2 numbers without using * operator
import java.io.*;
class DoWhile3{
 public static void main(String args[]) throws IOException{
BufferedReader kbd = new BufferedReader(new InputStreamReader(System.in));
System.out.println("Enter Any 2 Numbers");
 int a=Integer.parseInt(kbd.readLine());
 int b=Integer.parseInt(kbd.readLine());
 int i=1;
 int res=0;
  do{
 res = res + a;
    i++;
   }while(i<=b);
  System.out.println(" a*b = "+res);
  }
}
//wap to find the division of 2 numbers without using /,% operators and print quotient and remainder
```

Nested loop: Nested loop means writing a loop inside any other loop

```
// wap to print 1perfect numbers in the range of 1-1000
class NLoop1{
  public static void main(String args[]){
 for(int i=1;i<=1000;i++){
int j=1;
int sum=0;
while(j<=i/2){
  if(i%j==0){
   sum=sum+j;
  }
 j++;
 }
```

```
 if(sum==i){
   System.out.println(i);
  }
 }
}}
//wap to print following  output
        1       2       3       ....    10
        2       4       6       ....    20
        .
        .
        10      20      30      ....    100
class  NestedLoopDemo{
 public  static  void  main(String args[]){
 for(int i=1;i<=10;i++){
 for(int j=1;j<=10;j++){
  System.out.print(i*j+"   ");
  }
  System.out.println();
  }
 }
}
//wap to print following  design
*****
*****
*****
*****
*****
class  NestedLoopDemo{
 public  static  void  main(String args[]){
 for(int i=1;i<=5;i++){
  for(int j=1;j<=5;j++){
   System.out.print("*");
  }
  System.out.println();
 }
}
}
// wap to print following  design
*
**
***
****
*****
class  NestedLoopDemo{
 public  static  void  main(String args[]){
 for(int i=1;i<=5;i++){
  for(int j=1;j<=i;j++){
  System.out.print("*");
  }
```

```
  System.out.println();
   }
  }
 }
//wap to print following  design
1
22
333      //System.out.print(i);
4444
55555
//wap to print following  design
1
12
123      //System.out.print(j);
1234
12345
//wap to print following  design
5
55
555      //System.out.print("5");
5555
55555
//wap to print following  designs
1                                  1
2 3                                0 1
4 5 6                              1 0 1
7 8 9 10                           0 1 0 1
11 12 13 14 15                     1 0 1 0 1
//wap to print following shape
        *
       **
      ***
     ****
    *****
class  NestedLoopDemo{
  public  static  void  main(String args[]){
int  sp=15;
  for(int i=1;i<=5;i++){
  for(int k=1;k<=sp;k++){
   System.out.print(" ");
   }
  for(int j=1;j<=i;j++){
   System.out.print("*");
   }
   System.out.println();
   sp--;
   }
  }
 }
```

```
//wap to print following shape
      *
     ***
    *****    //  for(int i=1;i<=22;i=i+2)
   *******
  *********
//wap to print following shape
      *
     ***
    *****
   *******
    *****
     ***
      *
```

3. transferring statements

- transferring statements are the control statements which are used to transfer the control position from 1 location to another location
- In Java we have following 3 types of translators
    1.break
    2.continue
    3.return

1.break

-In java break statement is used only in 2 locations that is inside the switch-case or inside any loop. It means if we use break statement any another location then the code is invalid.

-break statement used in switch case to stop the continuity of switch-case and transfer the control from inside the switch-case to outside the switch-case by skipping the remaining statements of switch-case

-break statement used inside any loop to stop the continuity of the particular loop and transfer the control from inside the loop to outside the loop by skipping the remaining iterations of the loop

```
//WAP TO demo on break statement
class   BreakDemo{
 public  static  void  main(String args[]){
 for(int i=1;i<=10000;i++){
  if(i>10){
    break;
  }
 System.out.println(i);
 }
 System.out.println("HI");
 }
 }
```

Note:

It is always recommended to write break statement using if statement

2.continue

-In Java we use continue statement only in 1 location that is inside the any loop otherwise if we write continue in any other location then the code is invalid.

-continue statement used inside any loop to stop the continuity of current iteration and transfer the control from current iteration to next iteration by skipping by skipping the remaining statements available in the current iteration

```
//wap to demo on continue  statement
class   ContinueDemo{
```

```
 public  static  void  main(String args[]){
for(int i=1;i<=15;i++){
 if(i==2||i==7||i==11){
  continue;
  }
 System.out.println(i);
  }
 }
}
```

Note:

it is allways recommended to write continue statement using if statement.

3.return

return is a transferring statement which is used to stop the continuity of method execution.

```
//wap to demo on return statement
class   ReturnDemo{
 public  static  void  main(String args[]){
 System.out.println("Hi");
 if(10>2){
  return;
  }
  System.out.println("Bye!!!");
 }
 }
```

Note:

it is allways recommended to write return statement using if statement.

Arrays

-In our application Sometimes we need to store multiple values but if we want to store multiple values we need to declare multiple variables which leads to complexity in the program like length of the code increased, maintenance become complicated.

-If We want to resolve this problem we have to use Arrays concept. Arrays are the derived data types which are used to store multiple values using single variable

-We have following 2 types of Arrays

1. single dimensional array
2. multi dimensional array

1. single dimensional arrays

single dimensional arrays means we are storing multiple valuesin one dimension that is either horizontally or vertically.

declaration of SDA:

declaration of an array means we are deciding what is the name of the array and what type of values we are storing.

syntax

```
 datatype    arrayname[];
```

here

- datatype can be any primitive datatype or class type
- array name must be any java valid identifier
- array must be represented using []

Eg:

```
   int  arr[];
```

Rules

1. we can write a pair of square brackets [] after the variable or along with the data type

```
int  arr[];  ✓
int[]  arr;  ✓
int  []arr;  ✓
```

2. while declaring an array, array size must not be specified

```
int[]  arr1,arr2,arr3; ✓ // all arr1,arr2,arr3 will be declared as SDA
int arr1[],arr2,arr3; ✓ //only arr1 is SDA but arr2, arr3
                          are normal variable
int  arr1,[]arr2,arr3; ✗ CTE
int arr[5]; ✗ CTE
```

creation of SDA:

creation of SDA means providing the memory so that we can use in the application.

syntax:

```
datatype arrayname[] = new datatype[size];
                or
datatype arayname[];
arrayname = new datatype[size];
```

Eg:

```
int arr[] = new int[10];
```

Rules

1.while creating an array specifying array dimension or size is mandatory

2.array dimension or size must be byte/short/int/char but not long,float,double,boolean.

3.array dimension or size must not be negative but if we write size as negative value then the code is valid and compiled successfully but at run time it will throw a run time exception saying NegativeArraySizeException.

```
int  arr[]= new int[]; ✗ CTE: dimension is missing
int  arr[]= new int[10]; ✓
int  arr[]= new int[10.2]; ✗ CTE:
int  arr[]= new int['k']; ✓
int  arr[]= new int[101]; ✗ CTE:

int  arr[]= new int[0]; ✓
int  arr[]= new int[-2]; ✗ RTE: NegativeArraySizeException
int arr[] = new int[true]; ✗ CTE:
```

accessing elements of an Array

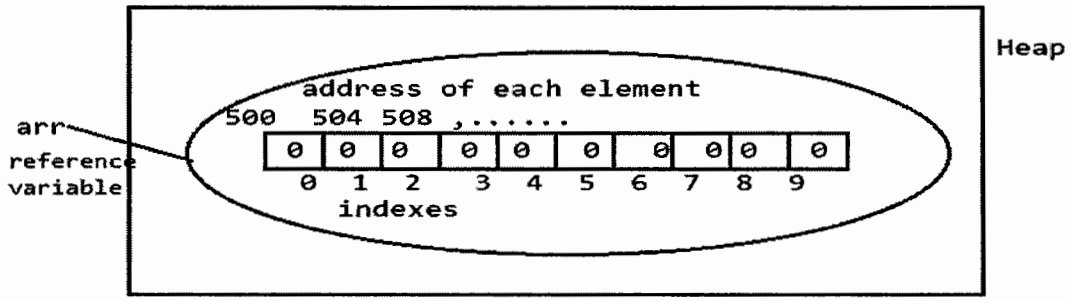
- To access the elements of an array we have to use a concept called indexes. (sub script reference)
- In java always array indexes are starts from 0 to size-1
- While accessing if we write array index <0 or >=size then it will throw a run time Exception saying ArrayIndexOutOfBoundsException.

```
int arr[] = new  int[10];
System.out.println(arr[0]);  ✓   // 0
System.out.println(arr[5]);  ✓   // 0
System.out.println(arr[7]);  ✓   // 0
System.out.println(arr[17]); ✗   //RTE:ArrayIndexOutOfBoundsException
System.out.println(arr[-7]); ✗   //RTE:ArrayIndexOutOfBoundsException
```

- in java arrays are maintained like an object.
- to create an array we have to use new operator.
- when an array is created the memory will be allcoated in heap memory and each location of an array will get memory and initialized with default values

Eg:

```
int arr[] = new  int[10];
```



-In arrays all elements are stored in continuous memory locations and address of each location is decided based on the size of the data type and index like follows,

address of I Element = startingaddress + index*datatypesize
= 500 + 0*4
= 500

address of II Element = startingaddress + index*datatypesize
= 500 + 1*4
= 504

address of III Element = startingaddress + index*datatypesize
= 500 + 2*4
= 508

.....

```
//wap to demo on SDA
class  SDArray1{
 public  static  void main(String args[]){
        int  arr[]  = new int[5];
    System.out.println(arr[0]);
    System.out.println(arr[1]);
    System.out.println(arr[2]);
```

```
      System.out.println(arr[3]);
      System.out.println(arr[4]);
     arr[0] = 10;
     arr[1] = 20;
     arr[2] = 30;
    arr[3] = 40;
    arr[4] = 50;
    System.out.println(arr[0]);
    System.out.println(arr[1]);
    System.out.println(arr[2]);
    System.out.println(arr[3]);
    System.out.println(arr[4]);
  }
}
//wap  to demo on SDA
class  SDArray2{
  public  static  void main(String args[]){
    int  arr[]  = new int[10];
   for(int i=0;i<10;i++){
      System.out.println(arr[i]);
    }
  for(int i=0;i<10;i++){
   arr[i]=(i+1)*10;
     }
  for(int i=0;i<10;i++){
      System.out.println(arr[i]);
   }
 }
}
//wap to read values from keyboard into an array and display total
import  java.io.*;
class  SDArray3{
public  static  void  main(String[] args)throws IOException{
   int  arr[] =  new  int[10];
BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
System.out.println("Enter Any 10 values");
  for(int i=0;i<10;i++){
     arr[i] =Integer.parseInt(br.readLine());
   }
System.out.println("Given Array values");
int total=0;
  for(int i=0;i<10;i++){
   System.out.println(arr[i]);
    total=total+arr[i];
   }
 System.out.println("total="+total);
 }
}
```

```
//wap to read array size from keyboard and finf small and big numbers in the given array
import java.io.*;
class SDADemo4{
 public static void main(String args[]) throws IOException{
BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
System.out.println("Enter Array size");
 int n = Integer.parseInt(br.readLine());
    int arr[] = new int[n];
System.out.println("Enter "+n+ "values");
for(int i=0;i<n;i++){
arr[i] = Integer.parseInt(br.readLine());
}
System.out.println("Given Array values");
for(int i=0;i<n;i++){
 System.out.println(arr[i]);
}
int big=arr[0];
int small=arr[0];
for(int i=0;i<n;i++){
 if (arr[i]>big){
big=arr[i];
 }
 if (arr[i]<small){
small=arr[i];
 }
}
 System.out.println("Big:"+big);
 System.out.println("Small:"+small);
  }
}
```

creating an array in a single line

-in java we can also create an array in single line

-if we want to initialize array with our own values then we have to use this concept.

-to create an array in single line we have to use following syntax

syntax:

```
    datatype arrayname[] = {list of values};
```

here

-creating an array in single line means we are declaring, creating and initializing an array at a time

-size of the array will be decided based on the number of values specified in between {}.

Eg:

```
 int iarr[] = {10,20,30,40,50};
 double darr[] = {1,2,3.4,5.6,7.8};
 char arr[] = {'g','g','g','r','r,'f','r'};
```

Rules

1. while creating an array in single line we must not specify the size, but size of the the array will be decided based the number of values specified in between {}.
2. List of values must be specified by using {}.

```
//wap to create an array in single line
class SDArrayDemo4{
```

```
  public static void main(String[] ar){
  double arr[] = {10,20,56.4,23.34,12.67,40};
System.out.println("\nArray  Values using normal for loop");
  for(int i=0;i<arr.length;i++){
   System.out.print(arr[i]+"   ");
  }
 System.out.println("\nArray  Values using foreach loop");
  for(double v:arr){
   System.out.print(v+"   ");
  }
 }
}
```

Note:

In java arrays are available like an object and there will be a predefined variable called " length " is available for every array which indicates the count of the number of elements available in the array (size of the array).

foreach loop (foreach is not a java keyword)

-foreach loop is a special loop introduced java 1.5 version which is used to access the elements of any array or any collection object.

-foreach loop is not a normal loop which is generally used to repeat the group of statements

-to write foreach loop again we have to write for keyword.

syntax:

```
for(declaration : arrayobj / collectionobj){
        //statements;
      }
```

rules

1. foreach loop must contain 2 sections separated by " : "
2. first section must be a declaration of variable to hold the each value of particular array or collection.
3. second section must be any array or collection object but it must not be any expression or value.

Multi Dimensional Arrays

-Multi dimensional arrays means storing multiple values in more than 1 dimension.

-The simplest form of Multi dimensional arrays is 2 dimensional array

syntax:

```
datatype arrayname[][] = new datatype[size1][size2];
                or
datatype arrayname[][];
arrayname = new datatype[size1][size2];
```

```
Declaration of 2 Dimensional array
-----------------------------------------
syntax:
     datatype   arrayname[][];
Eg:
      int  arr[][]; ✓
      int[][]  arr; ✓
      int  [][]arr; ✓
   int[]  arr[],arr1; ✓  arr is DD Array and arr1 is SD Array
    int[][]  arr1,arr2,arr3; ✓ all are DD Array
    int[]  arr1,[]arr2,arr3;  ✗ CTE:
    int[5][]  arr;  ✗CTE:
```

Creation of 2Dimesional Array

syntax:

```
datatype  arrayname[][] = new datatype[size1][size2];
```

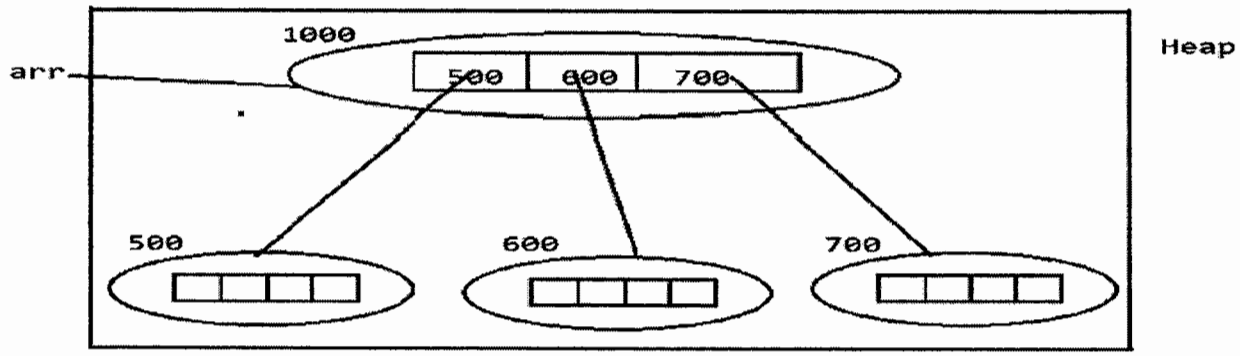
```
int  arr[][] = new int[][];   ✗
int  arr[][] = new int[5][];  ✓
int  arr[][] = new int[3][4]; ✓
int  arr[][] = new int[][5];  ✗
```

Rules

1. when we create a DD array specifying the size for first square bracket is mandatory for next square brackets it is optional.
2. here each square bracket will indicate one single dimensional array
3. here multi dimesional arrays are stored in multiple levels where all top level of arrays are containg addresses of next level of arrays only last level of arrays contains values

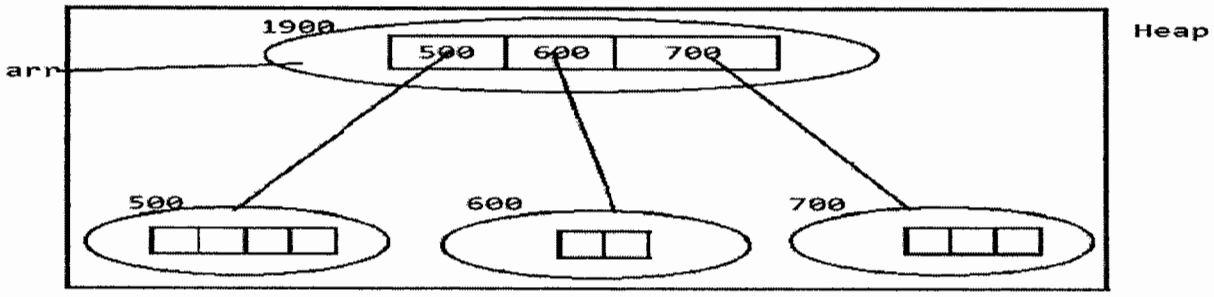
Eg1:

```
int arr[][] = new  int[3][4];
```



Eg2:

```
int arr[][] = new int[3][];
    arr[0]= new int[4];
    arr[1]= new int[2];
    arr[2]= new int[3];
```



in the last level we may have arrays with equal number of elements or non-equal number of elements.

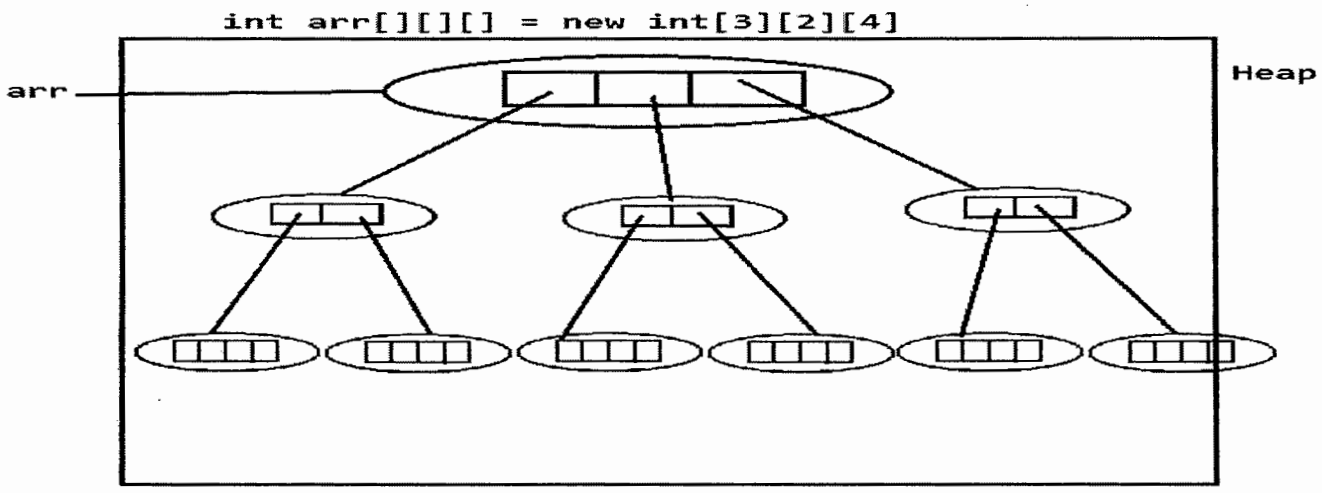
```
//wap to demo on DD Arrays which contains equal number of values in the last level of arrays
class DDArrayDemo1{
 public static void main(String[] args){
  int arr[][] = new int[3][4],k=1;
  for(int i=0;i<3;i++){
   for(int j=0;j<4;j++){
    arr[i][j]=k++;
    }
  }
 for(int i=0;i<3;i++){
  for(int j=0;j<4;j++){
   System.out.print(arr[i][j]+"  ");
  }
  System.out.println();
  }
  }
 }
//wap to demo on DD Arrays which contains non-equal number of values in the last level of
 arrays
class DDArrayDemo2{
public static void main(String[] args){
  int arr[][] = new int[3][],k=1;
    arr[0] = new int[4];
    arr[1] = new int[2];
    arr[2] = new int[5];
  for(int i=0;i<arr.length;i++){
   for(int j=0;j<arr[i].length;j++){
    arr[i][j]=k++;
   }
  }
for(int i=0;i<arr.length;i++){
  for(int j=0;j<arr[i].length;j++){
   System.out.print(arr[i][j]+"  ");
  }
  System.out.println();
  }
  }
 }
//wap to display DD Array using foreach loop
//wap to take 9 values into 3X3 DD array and print them in matrix format and print its
  transpose matrix format.
import java.io.*;
class DDADemo3{
 public static void main(String args[]) throws IOException{
BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
    int arr[][] = new int[3][3];
System.out.println("Enter Any 9 Numbers");
 for(int i=0;i<3;i++){
  for(int j=0;j<3;j++){
```

```
   arr[i][j] = Integer.parseInt(br.readLine());
   }
  }
System.out.println("Array Elements in Matrix Format");
  for(int i=0;i<3;i++){
   for(int j=0;j<3;j++){
    System.out.print(arr[i][j]+"  ");
   }
   System.out.println();
  }
            •
System.out.println("Array Elements in Transpose Matrix Format");
for(int i=0;i<3;i++){
  for(int j=0;j<3;j++){
  System.out.print(arr[j][i]+"  ");
  }
  System.out.println();
  }
 }
}
//wap to find addition and subtraction of given 2 3X3 matrices
import java.io.*;
class DDADemo4{
 public  static  void  main(String args[]) throws IOException{
BufferedReader  br = new BufferedReader(new InputStreamReader(System.in));
  int[][]  arr1 = new int[3][3],arr2 = new int[3][3],arr3 = new int[3][3],arr4 = new int[3][3];
System.out.println("Enter Any 9 Numbers into Matrix A");
 for(int i=0;i<3;i++){
  for(int j=0;j<3;j++){
   arr1[i][j] = Integer.parseInt(br.readLine());
  }
 }
System.out.println("Enter Any 9 Numbers into Matrix B");
  for(int i=0;i<3;i++){
  for(int j=0;j<3;j++){
   arr2[i][j] = Integer.parseInt(br.readLine());
   }
 }
for(int i=0;i<3;i++){
  for(int j=0;j<3;j++){
 arr3[i][j]  = arr1[i][j]+arr2[i][j];
 arr4[i][j]  = arr1[i][j]-arr2[i][j];
 }
}
System.out.println("Addition:");
for(int i=0;i<3;i++){
  for(int j=0;j<3;j++){
  System.out.print(arr3[i][j]+"  ");
  }
```

```
 System.out.println();
  }
System.out.println("Subtraction:");
for(int i=0;i<3;i++){
   for(int j=0;j<3;j++){
   System.out.print(arr4[j][i]+"  ");
   }
 System.out.println();
  }
 }
}
//wap to create 2 dimensional array in single line
class DDADemo5{
  public  static  void  main(String args[]) {
int  arr[][]={{89,56},{90},{90,45,45},{90,90,90,90}};
for(int i=0;i<arr.length;i++){
   for(int j=0;j<arr[i].length;j++){
   System.out.print(arr[i][j]+"  ");
   }
 System.out.println();
  }
 }
}
```

Note:

we can also write more than 2 dimensions for arrays hence multi dimensional arrays are also called as array of arrays



Hard coding

- Hard coding is a concept of providing values directly
- the main disadvantage of Hard coding is it always generate same output
- but if we want to generate any different output we have to perform several operations repeatedly like open the program, change the required values, save the program, compile and execute again.
- It is always recommend to remove the hard coding in the real time applications.
- To remove Hard coding we can use different methodologies like reading values from keyboard, reading values from a file, reading values from a text box, using command line arguments,....

command line arguments

- command line arguments means the values that we pass to main() method at command line or command prompt
- the main advantage of command line arguments is to remove hard coding.

```
//wap to demo on command line arguments
class Demo{
public static void main(String[] args){
int a,b,c;
a=Integer.parseInt(args[0]);
b=Integer.parseInt(args[1]);
c=a+b;
System.out.println("Addition="+c);
}
}
```

compilation

```
javac Demo.java
```

execution

```
java Demo 120 210
```

- we can enter multiple command line arguments which are separated by space
- we can enter multiple command line arguments of any type but it will take in the format of String.
- but if we want to get our original data or values we have to use " parsing "

parsing

parsing is a process of converting a string into required primitive type.

Eg:

```
Byte.parseByte();
Short.parseShort();
Integer.parseInt();
Long.parseLong();
Float.parseFloat();
Double.parseDouble();
Character.parseCharacter();  X – invali (this method is not available)
Boolean.parseBoolean();
```

- As a programmer we are responsible only for declaring String array in main() method.
- String array can be created by JVM and size will be decided by number of command line elements
- String array name can be anything

Eg:

```
String[] args
String data[]
String xyz[]
```

? why main() method takes String array only as parameter

If main() method takes integer array then we can enter only integer values as command line arguments or if main() method takes double array then we can enter only double values as command line arguments But If main() method takes String array then we can enter any type of value but we require parsing to convert into required type.

Eg:

```
class Cmd1{
  public static void main(String[] data){
System.out.println("No of Arguments: "+data.length);
 for(int i=0;i<data.length;i++){
  System.out.println(data[i]);
  }
```

```
  }
}
```

Eg:

```
class Cmd2{
   public static void main(String[]  data){
          int total=0;
           for(int i=0;i<data.length;i++){
             total = total + Integer.parseInt(data[i]);
         }
          System.out.println("total="+total);
  }
 }
```

String

-String is a predefined class available in java.lang package which is used to store group of characters.

-If we want to store group of characters we can use char array

Eg:

```
char  name[] = new char[10];
```

but here size is fixed, no predefined methods

-If we want to use String class we have to create an object for String class

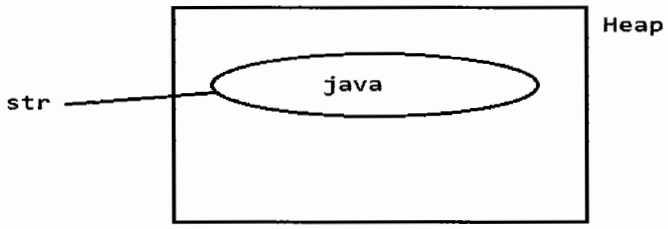
- In java 2 create an object for String class we have following 2 ways.

1. using new operator

Eg:

```
String str = new String("java");
```

when we create String object using new operator then String object is created inside the Heap memory.

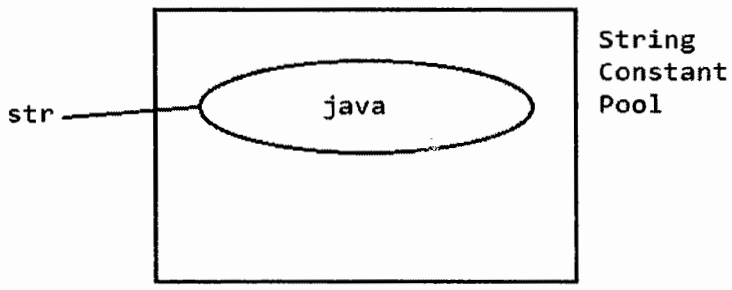


2. using string literal directly

Eg:

```
String str="java";
```

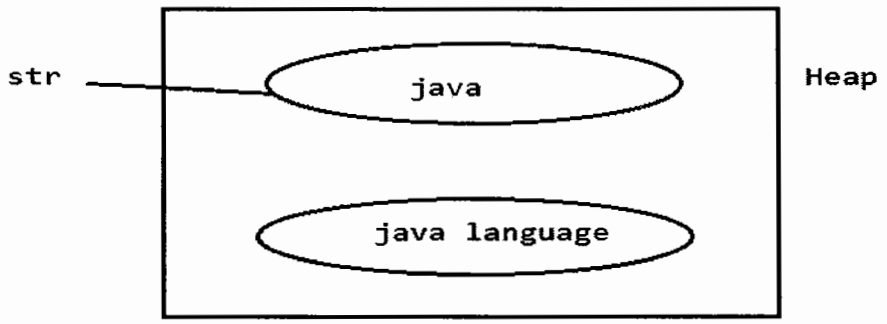
when we create String object using string literal directly then String object is created inside a special memory area called String Constant Pool.



immutable objects

String objects can be created either by using new operator or by using string literal directly in both the cases String objects are created as immutable objects.

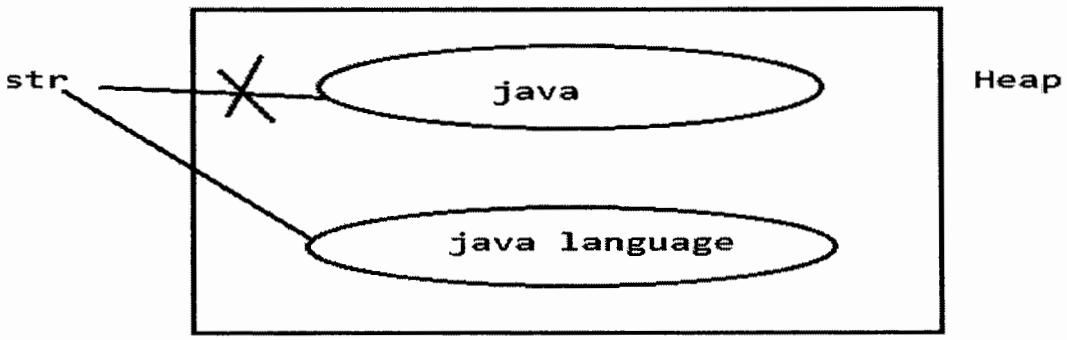
```
String   str = new String("java");
 System.out.println(str); //java
 System.out.println(str.concat(" language")); //java language
 System.out.println(str); //java
```



String objects are created immutable means, once the String is created its content never be changed instead of this it will create the new String object with the changed content

- But String reference variable can replace its objects with new object.

```
String str = new String("java");
System.out.println(str)//java
str = str.concat("language");
System.out.println(str)//java language
```

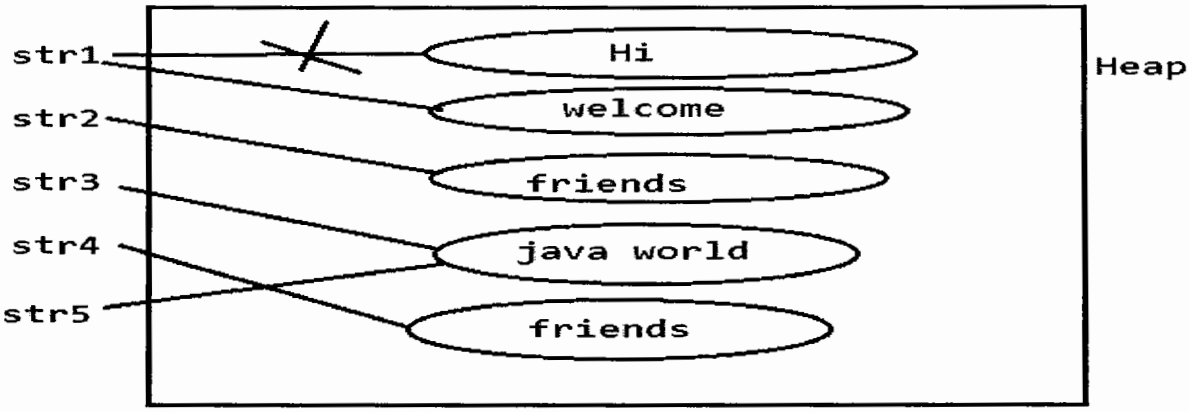


Difference b/w creating String object using new operator and String literal direclty

1. using new operator

- when we create String object using new operator then String object is created inside the Heap memory.
- while storing string object into Heap it never check whether the newly creating object is already existed or not simply it will create new object every time.

```
String  str1 = new String("Hi");
        str1 = new String("welcome");
String  str2 = new String("friends");
String  str3 = new String("java world");
String  str4 = new String("friends");
String  str5 = str3;
```



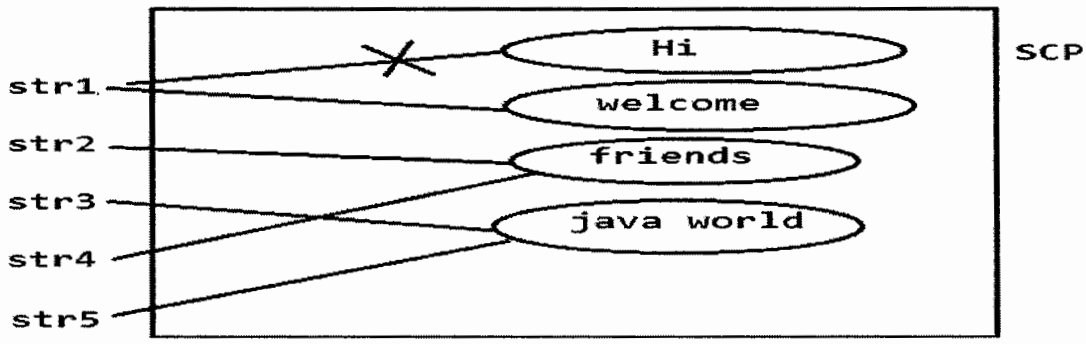
-Here (hi) Object any reference which is called un used object or unreferenced object that is ready for garbage collection.

- when there is any un used object is available in heap then it is the responsibility of garbage collector program to delete those unused objects.

2. using string literal directly

- when we create String object using string literal directly then String object is created inside String Constant Pool.
- while storing string object into String Constant pool it always check whether the newly creating object is already existed or not
- if object is already existed then reused the same object otherwise if not available it will create new object so that memory can be utilized efficiently.

```
String  str1 = "Hi" ;
        str1 = "welcome" ;
String  str2 = "friends" ;
String  str3 = "java world" ;
String  str4 = "friends" ;
String  str5 = str3;
```



- Here (hi) Object not having any reference which is called as unused object or Un referenced object.
- when there is any un used object is available in String constant pool then it is the responsibility of SCP itself to delete those un used objects.

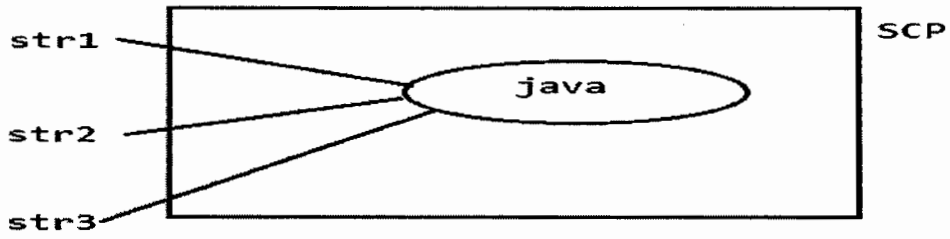
Note:

In Java a reference variable can refer to only one object at any time. But one object can referred by any number of reference variables.

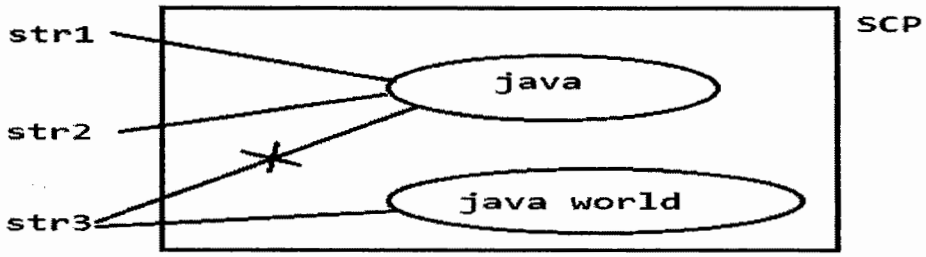
? Why String Objects are given as immutable objects

sometimes an object can be referred by multiple reference variables in this case if string objects are mutable objects then we change the content of object automatically other references get also modified so that string objects are given as immutable objects it means when ever any operation is done on strings it will create new object.

```
String  str1 = "java";
String  str2 = "java";
String  str3 = "java";
```

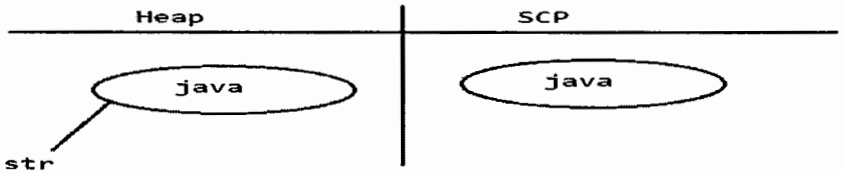



```
str3 = str3 +" world";
```

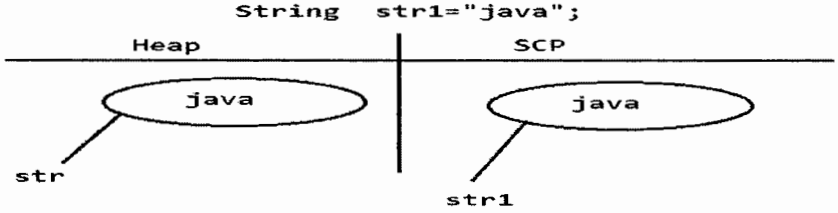


- when we create String object using new operator then String object is created inside the Heap memory and same copy of object is also created in SCP for re usability.

```
String  str = new String("java");
```




```
String  str1="java";
```



Methods of String class

1.int length()

this method returns the number of chars available in the given String.

Eg:

```
String str = new String("Java Language");
System.out.println(str); //Java Language
System.out.println(str.length()); //13
```

2.char charAt(int index)

- This method returns the character at specified index position available in the given String.
- This method especially used to access the string char by char

Eg1:

- accessing the string char by char

```
String str = new String("Java Language");
for(int i=0;i<str.length();i++){
  char ch = str.charAt(i);
  System.out.println(ch);
  }
```

Eg2:

```
String str = new String("Java Language");
System.out.println(str.charAt(0)); //J
System.out.println(str.charAt(5)); //L
System.out.println(str.charAt(20)); //RTE: StringIndexOutOfBoundsException
System.out.println(str.charAt(-1)); //RTE: StringIndexOutOfBoundsException
```

3. String concat(String)

This method is used to append the given 2 Strings

Eg:

```
String str = new String("Java");
System.out.println(str); //Java
System.out.println(str.concat(" Language"));//Java Language
System.out.println(str); //Java
```

4. String toLowerCase()

This method returns String in lowercase or small letters

5. String toUpperCase()

This method returns String in uppercase or capital letters

Eg:

```
String str = new String("Java Language");
System.out.println(str); //Java Language
System.out.println(str.toLowerCase()); //java language
System.out.println(str.toUpperCase()); //JAVA LANGUAGE
System.out.println(str); //Java Language
```

6. boolean equals(String)

This method used to compare the equality of the 2 Strings and it returns true if 2 strings are equal otherwise if not equal it returns false. this method consider the case of the strings.

7. boolean equalsIgnoreCase(String)

this method used to compare the equality of the 2 Strings same like equals() method but it will not consider the case of the strings.

Eg:

```
String str = new String("java");
System.out.println(str.equals("java")); //true
System.out.println(str.equals("Java")); //false
System.out.println(str.equalsIgnoreCase("java"));//true
System.out.println(str.equalsIgnoreCase("JAVA"));//true
System.out.println(str.equalsIgnoreCase("XYZ")); //false
```

8. int compareTo(String)

This method compares the 2 Strings based on ASCII code by considering the case.

if s1 > s2 then it returns +ve

if s1 < s2 then it returns -ve
if s1 == s2 then it returns 0

9. int compareToIgnoreCase(String)

This method also compares the 2 Strings based on ASCII code without considering the case

Eg:

```
String str = "c";
System.out.println(str.compareTo("a")); // +ve
System.out.println(str.compareTo("f")); // -ve
System.out.println(str.compareTo("C")); // -ve
System.out.println(str.compareToIgnoreCase("C")); //0
```

10. boolean startsWith(String)

This method returns true when the given string starts with specified string otherwise it returns false (it always consider the case)

11. boolean endsWith(String)

This method returns true when the given string ends with specified string otherwise it returns false (it always consider the case)

Eg:

```
String str = "Java Language";
System.out.println(str.startsWith("Jav")); //true
System.out.println(str.startsWith("Javx")); //false
System.out.println(str.endsWith("age"));//true
System.out.println(str.endsWith("agex"));//false
```

12. int indexOf(char) or int indexOf(String)

This method returns index of the first occurrence of specified character. It always begin searching from index 0.

13. int indexOf(char,int index)

This method returns index of the specified character. It will begin searching the character from specified index.

14. int lastIndexOf(char) or int lastIndexOf(String)

This method returns index of the last occurrence of specified character.

Eg:

```
String str = "Java Language";
System.out.println(str.indexOf('a')); // 1
System.out.println(str.indexOf('a',str.indexOf('a')+1)); //3
System.out.println(str.lastIndexOf('a')); //10
System.out.println(str.indexOf('p')); //-1
System.out.println(str.lastIndexOf('q')); //-1
System.out.println(str.lastIndexOf('A')); //-1
System.out.println(str.lastIndexOf("Lan")); //5
System.out.println(str.lastIndexOf("an")); //6
System.out.println(str.lastIndexOf("Lanx")); //-1
```

Note:

-all these 3 methods are case-sensitive

-if the specified String or character not available all these 3 methods returns -1

15. String substring(int index)

This method used to capture the part of the string from the given string. It will capture from specified index to end of the string.

16. String substring(int index,int offset)

This method also used to capture the part of the string from the given string. It will capture from specified index to specified offset (position of the character).

- index begin from 0 to size-1 - but offset will begin from 1 to size

Eg:

```
String str = "Java Language Is Good";
System.out.println(str.substring(5)); // Language is Good
System.out.println(str.substring(0,4)); // Java
System.out.println(str.substring(5,8)); // Lan
```

17.String replace("oldstring","newstring")

This method replaces the specified chars of the string with newly specified string of characters if available otherwise it wont replace anything and returns the same string.

Eg:

```
String str = "Java Language";
System.out.println(str.replace("Java","VB"));//VB Language
System.out.println(str.replace("java","VB")); //Java Language
System.out.println(str.replace("xyz","VB")); //Java Language
```

18. String trim()

This method removes extra spaces given before the text and after the text.

Eg:

```
String str = "    Java    Language    ";
System.out.println("HI"+str+"OK");
System.out.println("HI"+str.trim()+"OK");
```

Rules

1.we never use comparison operators like <,>,<=,>= to compare to Strings directly. But if we want to achieve this requirement we have to use compareTo() method of String class

2.but we can use comparison operators or equality operators like ==,!= to Compare 2 Strings

```
String str1="java";
String str2="java";
```

```
if(str1>str2){
}
```

```
if(str1<str2){
}
```

```
if(str1>=str2){
}
```

```
if(str1<=str2){
}
```

```
if(str1==str2){
}
```

```
if(str1!=str2){
}
```

.equals() method and == operator

-.equals() method always check the equality of the content of given 2 strings. (content comparison)

- but == operator always used to compare the address of 2 reference whether they are referring to the same object or not (address comparison)

Eg:

```
String str1 = new String("java");
String str2 = new String("java");
String str3 = "java";
String str4 ="java";
String str5=str1;
//content comparision
System.out.println(str1.equals(str2)); //true
System.out.println(str1.equals(str3)); //true
System.out.println(str1.equals(str4)); //true
```

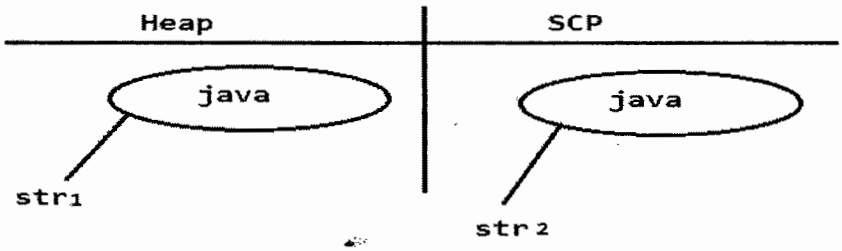
```
System.out.println(str1.equals(str5)); //true
//address comparision
System.out.println(str1==str2); //false
System.out.println(str1==str3); //false
System.out.println(str1==str4); //false
System.out.println(str1==str5); //true
System.out.println(str3==str4); //true
```

19.String intern()

This method is used to return the corresponding object of the string reference which is available in SCP.

Eg:

```
class StringTest{
  public static void main(String[] data){
   String str1 = new String("Java");
   String str2 = str1.intern();
   System.out.println(str1); //java
   System.out.println(str2); //java
   System.out.println(str1==str2); //false
   }
}
```



Concatenation operator(+)

-if we use + operator between numerical data type values then it will perform arithmetical addition operation

-but if we use + operator between 2 strings or between string and any primitive then it will do Concatenation operation

Eg:

```
String str =new String("java");
System.out.println(str+1+2+3+4);//java1234
System.out.println(str+1+2+3+4+true);//java1234true
System.out.println(1+2+3+4+str);//10java
//System.out.println(true+1+2+3+4+str);//invalid
System.out.println(1+2+str+3+4);//3java34
System.out.println(str+1+(2+3)+4);//java154
System.out.println(str+(1+2+3+4));//java10
```

Method chaining

Method Chaining is a process of calling string methods continuously using .operator.

```
String str = new String("Java");
System.out.println(str.concat(" Language").toUpperCase()
.replace("JAVA","VB").substring(3).length());
```

```
System.out.println(str.concat(" Language").toUpperCase()
.replace("JAVA","VB").substring(3).length().toLowerCase());
```

```
//wap to check whether the given String is palindrome or not
import java.io.*;
class String1{
public static void main(String[] args) throws IOException{

        BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
      System.out.println("Enter Any String");
        String str1 = br.readLine();
         String str2="";
      for(int i=0;i<str1.length();i++){
          char ch = str1.charAt(i);
          str2 = ch+str2;
        }
      if(str1.equals(str2)){
      System.out.println("It is a Polindrome String");
      }
      else{
      System.out.println("It is Not a Polindrome String");
   }
  }
 }
//wap to count no.of Words in a String
import java.io.*;
class String2{
public static void main(String[] args) throws IOException{

      BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
      System.out.println("Enter Any String");
        String str = br.readLine();
       str=str.trim();
      int c=0;
      String temp="";
    for(int i=0;i<str.length();i++){
          char ch = str.charAt(i);
       if(ch==' '){
       if(temp.length()>0){
        c++;
        temp="";
        }
       }
      else{
        temp=temp+ch;
         }
   }
  System.out.println("No.of Words="+(c+1));
  }
 }
```

Assignments

//wap to count the number of alphabets, number of digits, number of special chars in the given string

// wap to count the number of vowels in the given string

//wap to print string like follows

given: Hi How Are You

o/p: iH woH erA uoY

//wap to convert the given string into lowercase and uppercase without usig toLowerCase() & toUpperCase() methods

StringBuffer

- StringBuffer is another predefined class available in java.lang package which is also used to store group of characters same like String class.

-But String class objects are immutable objects where StringBuffer Objects are called mutable objects.

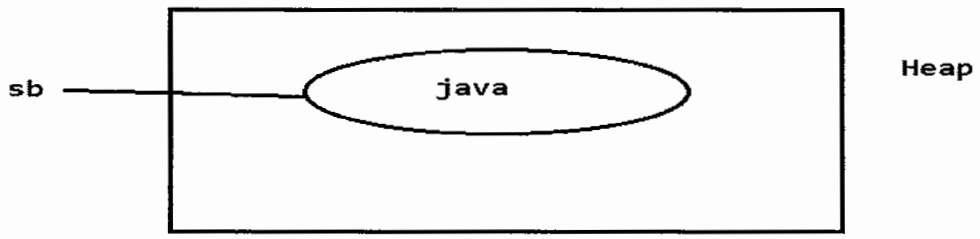
-StringBuffer Objects are mutable objects means the content of StringBuffer can be changed any number of times.

-we have only one way to create StringBuffer object that is using new operator

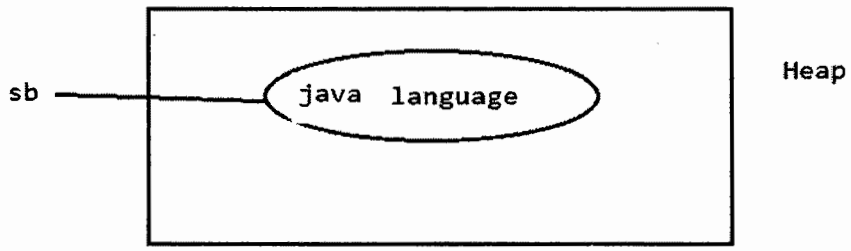
Eg:

StringBuffer sb = new StringBuffer("java");

```
StringBuffer    sb = new StringBuffer("java");
  System.out.println(sb); //java
```

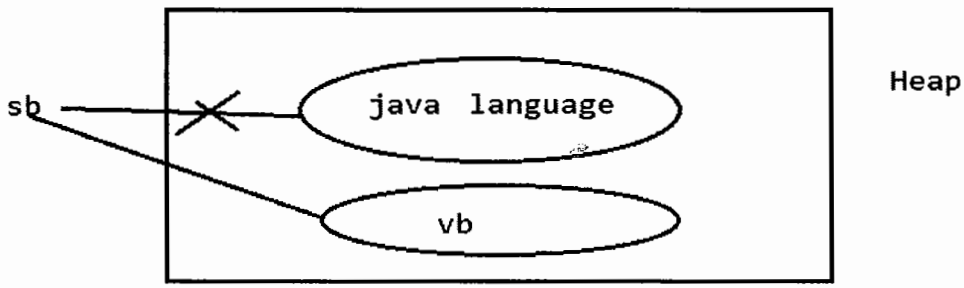



```
System.out.println(sb.append(" language")); //java language
    System.out.println(sb);//java language
```



- We can also replace the StringBuffer object with new StringBuffer Object

```
sb = new StringBuffer("vb");
```



Methods of StringBuffer

1. int length()

this method returns the count of the number of chars available in the StringBuffer.

2. StringBuffer append(xxxx):

this method is used to append the specified content with the existing content.

Eg:

```
StringBuffer sb = new StringBuffer("java");
System.out.println(sb); //java
System.out.println(sb.append(" Language"));//java Language
System.out.println(sb.append(7.0));//java Language7.0
System.out.println(sb.append(10));//java Language7.010
System.out.println(sb.append(true));//java Language7.010true
System.out.println(sb);//java Language7.010true
```

3.StringBuffer insert(int index,xxxx)

this method insert the specified content at specified position in the String Buffer.

Eg:

```
StringBuffer sb = new StringBuffer("java");
System.out.println(sb); //java
System.out.println(sb.append(" Language")); //java Language
System.out.println(sb.insert(4," Is Good ")); //java is Good Language
System.out.println(sb.insert(0,7.0)); //7.0java is Good Language
System.out.println(sb); //7.0java is Good Language
```

4. StringBuffer deleteCharAt(int index)

this method delete character at specified index and returns StringBuffer.

5. StringBuffer delete(int index,int offset)

this method delete character at specified index to specified offset and returns StringBuffer.

Eg:

```
StringBuffer sb = new StringBuffer("java Language");
System.out.println(sb); //java Language
System.out.println(sb.deleteCharAt(1)); //jva Language
System.out.println(sb.delete(4,9));//jva age
System.out.println(sb);//jva age
```

6. String substring(int index)

this method capture the substring from specified index to end of the string in the given StringBuffer

7. String substring(int index,int offset)

this method capture the substring from specified index to specified offset of the string in the given StringBuffer

Eg:

```
StringBuffer sb = new StringBuffer("java Language");
System.out.println(sb); //java Language
System.out.println(sb.substring(5));//Language
System.out.println(sb.substring(5,8));//Lan
```

8. StringBuffer replace(int index,int offset, String)

this method replace the String in the StringBuffer at specified positions.

Eg:

```
StringBuffer sb = new StringBuffer("java Language");
System.out.println(sb); //java Language
System.out.println(sb.replace(0,4,"VB"));//VB Language
```

```
        System.out.println(sb); //VB Language
```

9.StringBuffer_reverse()

this method reverse the content of given StringBuffer

Eg:

```
        StringBuffer  sb = new  StringBuffer("java Language");
        System.out.println(sb); //java Language
        System.out.println(sb.reverse()); //egaugnaL avaj
        System.out.println(sb); //egaugnaL avaj

//wap to find given string palindrome or not using StringBuffer
import  java.io.*;
class    StringBuffer1{
 public  static  void  main(String[]  ar) throws IOException{
        BufferedReader  br = new BufferedReader(new InputStreamReader(System.in));
        System.out.println("Enter Any String");
          String    str1 = br.readLine();
        StringBuffer  sb = new StringBuffer(str1);
         sb.reverse();
        String  str2 = new String(sb);
        if(str1.equals(str2)){
         System.out.println("It is a Palindrome String");
        }
        else{
         System.out.println("It is Not a Palindrome String");
         }
    }
 }
```

StringBuilder (java 1.5)

- StringBuilder is another predefined class which is also used to store group of chars.
- StringBuilder is exatly similar to StringBufffer but StringBuilder class is not a synchronized class where StringBuffer class is synchronized class.

javap

- This java command mainly used to display the profile of any predefined class or user defined class. Profile means it will display the list of variables and methods that particular class contains.

syntax

```
        javap  packagename.classname
```

Eg:

```
        javap  java.lang.String
        javap  java.lang.StringBuffer
        javap  Demo
```

Java API Documentation

- Java API Documentation is also used to see profile of any predefined class in detail.
- Java API Documentation we can freely download from internet.

OOPS

If we want to develop any application we can develop by following any one of the following systems.

1. procedure-oriented programming systems
2. object-oriented programming systems

1. procedure-oriented programming systems

If we want to develop an application based on this procedure-oriented programming systems we take the help of functions or procedures

Eg:

C,COBOL,PASCAL,FORTRAN,....

drawbacks

1. In this system we create multiple functions which will increase the code side so that maintenance will be complicated and debugging will become complicated
2. in this system we mainly concentrate on operations or functions but not on data which is very important for our application
3. in this system data is open and globally available
4. there is no data security in this system
5. unctions can not be expanded

2. object-oriented programming systems

-If we want to develop an application based on this procedure-oriented programming systems we take the help of objects and classes

-this system can resolve the problems of previous system that is procedure oriented system.

-the concepts of oops are very strong and suitable for any large and complicated projects

- oops concepts are developed based on human life so that we can easily understand and implement in the application.

Eg:

java, c++, .net, small talk, ...

Features of OOPS

1. encapsulation

- Encapsulation is a process of grouping data (variables) and operations (methods) as a single unit
- The main advantage of encapsulation is data hiding
- We can achieve encapsulation using classes
- Using Encapsulation we can also improve the maintenance of the application by isolating

2. abstraction

- abstraction means hiding the internal details and providing the use full information
- using abstract classes and interfaces we can achieve this abstraction
- the main advantage of abstraction is data security

3. inheritance

- inheritance means taking the properties of one class into another class
- the main advantage of inheritance is reusability

4.polymorphism

- polymorphism means defining one entity with multiple behaviors
- the main advantage of polymorphism is flexibility because using one entity we an perform multiple operations

classes

-A class is a Plan or model using which we can create an object

-A class contains a set of variables and methods together called as members of the Class

syntax:

```
class ClassName{
    datatype  var1;
    datatype  var2;
    datatype  var3;
    .....
    returntype  methodname1(list of parameters){
    }

    returntype  methodname2(list of parameters){
    }
    returntype  methodname3(list of parameters){
    }
    .....
}
```

objects

- An object is a physical entity of a class which is used to access the members of the class
- An object is also called instance of the class
- Object contains a set of properties and actions
- Properties nothing but variables of the class which hold values and describe about of the object.

-Actions are nothing but methods of the class which can tell us about functionality of the object what it can do.

syntax:

```
ClassName  referencevar = new  ClassName();
```

- once object is created the memory allocated in Heap memory

Eg:

```
class  Student{
        int rno;
        String name;
        double fee;
        void display(){
        int a=10;
        System.out.println("RNO:"+rno);
        System.out.println("NAME:"+name);
        System.out.println("FEE:"+fee);
        }
        public static void main(String  args[]){
        /*
          Student  s = new  Student();
              s.display()
        //accessing variables using reference
          s.rno=7;
          s.name="sachin";
          s.fee=1200.00;
        System.out.println("rno="+s.rno);
        System.out.println("name="+s.name);
        System.out.println("fee="+s.fee);
```

```
        //local variables are not part of object
        //System.out.println("a="+s.a);  // X-invalid
        s.display();
        */
         Student  s;
          s = new  Student();
          s.display();
        //anonymous objects
        /* new Student();  //valid but no output*/
        /*(new Student()).display();//valid & student data displayed*/
        /*
        RTE:NullPointerException
        Student s=null;
        s.display();
        */
         }
}
```

instance variables

- The variables declared in side class and outside the methods then these variables are called as instance variables. it means they are part of the object.
- when we declare instance variables and if we don't provide any value then they automatically initialized with default values.

data type	default value
byte	0
short	0
int	0
long	0
float	0
double	0
char	space
boolean	FALSE
String	null
StringBuffer	null
Student	null
any reference	null

- instance variables are used in the class to store the data which are also called as properties.

instance methods:

- In a class we can declare methods to perform some operations.
- Methods are nothing but actions that are done by the particular object.
- In a class we can declare any number of methods including main() method but JVM always invoke only main() method
- But if we want to invoke other methods which are defined by the programmer then the programmer is only responsible to invoke these methods explicitly.

Note:

-In Java declaring the variables is dynamic it means we can declare the variables any where inside the program

-Java language is called as strongly type language because it always check for the variables whether they declared or not and type of the value that we are storing is correct or not

User defined methods

- in java we can also declare our own methods
- every method contains following 2 parts
  1.method declaration/ method prototype / method header
  2.method body/method definition/method implementation

1. method declaration/ method prototype / method header

method declaration means we are going to decide what is the name of the method , what are the parameters it will take and what kind of value is return by the method.

syntax:

```
returntype   methodname(list of parameters);
```

returntype

-returntype is data type that indicates what type of value is return by the particular method.

-returntype can be any primitive type or array type or reference type

-if method does not return any value then return type must be specified using a java keyword called " void ".

-specifying returntype is mandatory

methodname

-To identify and access the method there is a suitable name is given for a method which is called as method name.

-a methodname can be any valid java identifier.

-specifying method name is mandatory

list of parameters

-parameters are the variables that will receive the values that are passed into the. particular method on which data method will perform the operations.

- we can write 0 or more number of parameters of any primitive type or array type or reference type
- specifying parameters is optional.

2. method body/method definition/method implementation

-method definition means we are going to write the group of statements that are executed by the method.

-a method definition can be written in between a pair of curry braces

syntax:

```
returntype   methodname(list of parameters){
        //statements;
        return value;
        }
```

-here we can write 0 or more number of statements in between the pair of curly braces.

-when we write 0 statements then it is called as null implementation

-if the return type is specified as other than void then we must return a value from our method using java keyword called " return ".

syntax:

```
return value;
```

- the datatype of the value we return must be match with the datatype that we specify as return type.
- but if return type specified as void then we must not write any return value statement.
- In java we can create any number of methods which are in any of the following 4 combinations of methods
  1. method without return type, without parameters
  2. method without return type, with parameters
  3. method with return type, without parameters
  4. method with return type, with parameters

Method Invocation

Method invocation is a process of calling the method and executing the logic written within the method.

syntax:

methodname(list of parameters);

1. To design a method without return type, without parameters

```
class Method1{
//method declaration and definition
  void  add(){
  int a,b,c;
  a=10;
  b=20;
 c=a+b;
System.out.println("Addition="+c);
   }
public static void main(String args[]){
 Method1 m = new Method1();
  m.add(); //method invocation
  m.add();
 }
}
```

2. To design a method without return type, with parameters

```
class Method2{
 void  add(int x,int y){
  int a,b,c;
  a=x;
  b=y;
 c=a+b;
 System.out.println("Addition="+c);
}
public static void main(String args[]){
 Method2 m = new Method2();
 m.add(67,56);
 m.add(789,567);
 int p=10,q=90;
 m.add(p,q);
}
}
```

3. To design a method with return type, without parameters

```
class  Method3{
public static void main(String args[]){
 Method3 m =  new Method3();
 int res = m.add();
System.out.println("Addition1="+res);
System.out.println("Addition2="+m.add());
}
int add(){
int a,b,c;
a=10;
b=20;
```

```
c=a+b;
return c;
}
}
```

4.To design a method with return type, with parameters

```
class Method4{
public static void main(String args[]){
 Method4 m = new Method4();
 int res = m.add(78,90);
System.out.println("Addition1="+res);
System.out.println("Addition2="+m.add(90,34));
}
int add(int x,int y){
int a,b,c;
a=x;
b=y;
c=a+b;
return c;
}
}
```

//wap to design a method with array as parameter

```
class Method5{
void showArray(int ar[]){
  for(int v:ar){
  System.out.print(v+"  ");
   }
  System.out.println();
}
public static void main(String args[]){
   Method5 m = new Method5();
 int arr[] = {89,89,78,67,56,56,78};
    m.showArray(arr);
    m.showArray(new int[]{56,56,67});  //anonymous array
   // m.showArray(10,20,30,40); -invalid
 }
}
```

//wap to design a method with array as return type

```
class Method6{
int[] getArray(){
int arr[] = {89,89,78,67,56,56,78};
return arr;
}
public static void main(String args[]){
   Method6 m = new Method6();
  int ar[]= m.getArray();
 for(int v:ar){
  System.out.print(v+"  ");
   }
 }}
```

//wap to design a method with class or reference as a parameter.

```
class Employee{
int empno;
String ename;
}
class Method7{
void showEmployee(Employee e){
 System.out.println("EMPNO:"+e.empno);
 System.out.println("ENAME:"+e.ename);
}
public static void main(String args[]){
   Method7 m = new Method7();
   Employee eee = new Employee();
    eee.empno=7;
    eee.ename="sehwagh";
   m.showEmployee(eee);
 }
}
```

//wap to design a method with class or reference as a return type.

```
class Employee{
int empno;
String ename;
}
class Method8{
Employee   getEmployee(){
   Employee eee = new Employee();
    eee.empno=7;
    eee.ename="sehwagh";
 return eee;
}
public static void main(String args[]){
   Method8 m = new Method8();
  Employee e = m.getEmployee();
 System.out.println("EMPNO:"+e.empno);
 System.out.println("ENAME:"+e.ename);
}
}
```

Method Recursion

- Method Recursion is a process of calling a method within itself
- Method Recursion we mainly use for achieving reverse algorithm

  Eg:

  games, iterating any data structures,....

Note:

- In Method Recursion there is a chance to enter into infinite loop and it wil throw a Runtime exception saying StackOverFlowError

//wap to demo on method recursion

```
class Method9{
void show(int n){
if(n==0){
```

```
return;
}
System.out.println(n);
 show(n-1);
}
public static void main(String ar[]){
Method9 m = new Method9();
  m.show(10);
}
}
```
//wap to calc factorial of given number using method recursion
```
class Method10{
int fact(int n){
if(n==1){
return 1;
}
else{
return n*fact(n-1);
}
}
public static void main(String ar[]){
Method10 m = new Method10();
 int res = m.fact(5);
System.out.println("5!="+res);
}
}
```
//wap to display all armstring numbers using methods
```
class Method11{
boolean isArmStrong(int n){
   int r,sum=0,m=n;
    while(n>0){
      r=n%10;
     sum=sum+r*r*r;
      n=n/10;
      }
    if(sum==m){
     return true;
    }
   else{
     return false;
    }
   }
public static void main(String ar[]){
Method11 m = new Method11();
for(int i=1;i<=1000000;i++){
   if(m.isArmStrong(i)==10){
    System.out.println(i);
   }
}}}
```

Constructors

-when we declare instance variables and not initialized with any value then they automatically initialized with default values

-but if we want to initialize instance variables withour own values then we can initialize instance variables in following 2 locations,

1. At the time of declaration 2. using contructor

constructors

-constructor is also a method which is especially designed to initialize instance variables.

-to create a constructor we have to follow following rules

Rules

1. Constructor name must be same as ClassName
2. constructor can not take any return type but if we write any return type then the code is valid but it is considered as normal method
3. we can not return any value from the constructor
4. constructors take parameters
5. Based on the number of parameters constructors are classified into following 2 types,

1. 0 parametrized constructor

if we declare any constructor without any parameters then it is called as 0 parametrized constructor.

syntax:

```
class ClassName{
  ClassName(){
    //statements;
  }
}
```

2. parametrized constructor

if we declare any constructor withparameters then it is called as parameterized constructor.

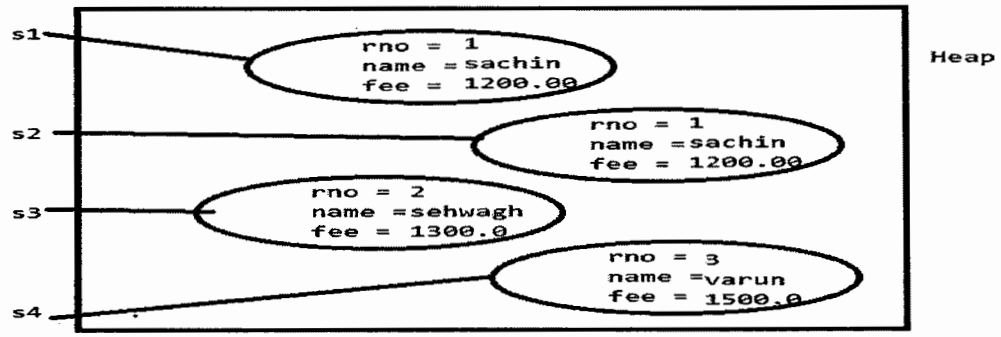
syntax:

```
class ClassName{
   ClassName(parameters){
    //statements;
   }
}
```

6. constructors can be invoked and accessed at the time of object creation.

```
// wap to demo on constructors
class Student{
 int rno;
 String name;
 double fee;
 /*0 parameterized constructor */
Student(){
 rno=1;
 name="sachin";
 fee=1200.0;
}
 /* parameterized constructor */
Student(int rn,String nm,double fe){
 rno=rn;
 name=nm;
 fee=fe; }
```

```
/*normal method*/
 void  display(){
   System.out.println(rno+"\t"+name+"\t"+fee);
  }
}
class Constructor1{
 public static void main(String args[]){
   Student s1 = new Student();
    s1.display();
   Student s2 = new Student();
    s2.display();
   Student s3 = new Student(2,"sehwagh",1300);
    s3.display();
   Student s4 = new Student(3,"dhoni",1500);
    s4.display();
  }
}
```



1. Memory allocated for instance variables at the time of declaration of object
2. Memory allocated for instance variables within the Heap memory area of JVM
3. Memory allocated for instance variables 1 time for Every object.
4. Different copy of instance variables given to every object of the particular class. It means the data changed in one object can not effect the data of other object.

Note:

-If we want to store same data in all the objects initially then we have to initialize instance variables at the time of declaration and we can also use 0 parametrized constructor

-If we want to store different data in all the objects initially then we have to initialize instance variables using parametrized constructor

-In java we can not create any object for any class without using constructor.

-In java every class both user defined or predefined classes contains a constructor whether we specify or not.

```
class Demo{
   }
```
compile →
```
class Demo{
  Demo(){
   }
}
```

```
class Demo{
  Demo(){
   }
}
```
compile →
```
class Demo{
  Demo(){
   }
}
```

```
class Demo{
 Demo(int x){
   }
}
```
compile →
```
class Demo{
 Demo(int x){
   }
}
```

- compiler always generate 0 parameterized constructor only when we dont write any constructor
- it is always recommended for creating 0 parameterized constructor when we specify parameterized constructor

this keyword

" this " is java keyword/reference which is referring to current class instance or object so that can access the all members of current class using this "this" keyword with in the same class.

Eg:

```
class Employee{
int empno;
String ename;
double salary;
Employee(int empno,String ename,double salary){
this.empno=empno;
this.ename=ename;
this.salary=salary;
this.display();
}
void display(){
 System.out.println("EMPNO:"+empno);
 System.out.println("ENAME:"+ename);
 System.out.println("SALARY:"+salary);
}
}
class Constructor2{
public static void main(String ar[]){
Employee e = new Employee(1,"sehwagh",120000.0);
 }
}
```

static keyword

If we declare instance varables they will get memory 1 time for 1 object, it means if we create multiple objects then memory allocated for multiple times. But if we want to allocate the memory for instance variables only for 1 time then we have to declare instance variables using java keyword called "static"

static variables

-The variables declared in side class and outside the methods with static keyword then these variables are called static variables.

syntax:

```
static datatype var1,var2,...;
```

-Memory allocated for static variables at the time of Class Loading

-Memory allocated for static variables inside method area of JVM

-Memory allocated for static variables 1 time for Entire Class

-Same copy of static variables will be given to all the different objects of the particular class. It means if one object change the values of then other objects get modified automatically.

-Simply static variables are called as class variables

//wap to demo on static variables

```
class Student{
 int rno;
String name;
 double fee;
```

```
static  String cname="inetsolv";
Student(){
rno=1;
name="sachin";
fee=1300.0;
}
Student(int r,String n,double f){
rno=r;
name=n;
fee=f;
}
void  display(){
System.out.println("Rno:"+rno+"\t"+"Name:"+name+"\t"+"Fee:"+fee+"\tCname:"+cname);
}
}
class StaticDemo1{
public  static  void  main(String  args[]){
Student s1=  new Student();
  s1.display();
Student s2=  new Student();
  s2.display();
Student s3=  new Student(3,"sehwagh",1400.0);
  s3.display();
Student s4=  new Student(4,"kohli",1000.0);
  s4.display();
s2.rno=8;
s2.name="dhavan";
s2.fee=700;
s2.cname="InetSolv Infotech";
 s1.display();
 s2.display();
 s3.display();
 s4.display();
}
}
```

static methods

we can also declare a method using static keyword which are called as static methods.

syntax:

```
 static  returntype  methodname(parameters){
    //statements;
    return  value;
   }
```

Note:

1.in a class we can declare instance variables and instance methods which are together called as instance members which can be accessible only by using object of the class.

2.in a class we can declare static variables and static methods which are together called as static members which can be accessible either by using object of the class or by using class Name directly.

```
//wap to demo on static variables and static methods
class  Demo{
static int a=10;
static  int b=20;
static  void show(){
System.out.println("a="+a+"\t"+"b="+b);
}
}
class StaticDemo2{
public static void main(String  args[]){
//accessing static members using instance or object of the class
 Demo  d = new  Demo();
 System.out.println("a="+d.a+"\t"+"b="+d.b);
 d.show();
//accessing static members using  ClassName directly
System.out.println("a="+Demo.a+"\t"+"b="+Demo.b);
 Demo.show();
}
}
```

Note:

- we can access static members of the class either by using ClassName directly / by using object,but it is always recommended that accessing static members using ClassName only because object may be available or may not be available.

? why should we use static variables

when we want to allocate the memory for instance variables only for 1 time for entire class and they can be shared between multiple objects of the particular class. then we need to declare the particular variable as static variable, we can also access static variables directly by using ClassName.

? why should we use static methods

when we want to access any method directly by using ClassName then we have to declare particular method as static method.

Rules

1. An instance method can access both instance members and static members.
2. But a static method can access only static members directly. but if we want to access instance members inside the static methods we have to use an object of the particular class.

```
class  Demo{
 int a=10;
static int b=20;
void  display(){
  System.out.println("a="+a);
  System.out.println("b="+b);
 }
static  void  show(){
//System.out.println("a="+a); - invalid
  Demo  d = new  Demo();
  System.out.println("a="+d.a);
  System.out.println("b="+b);
 }
}
```

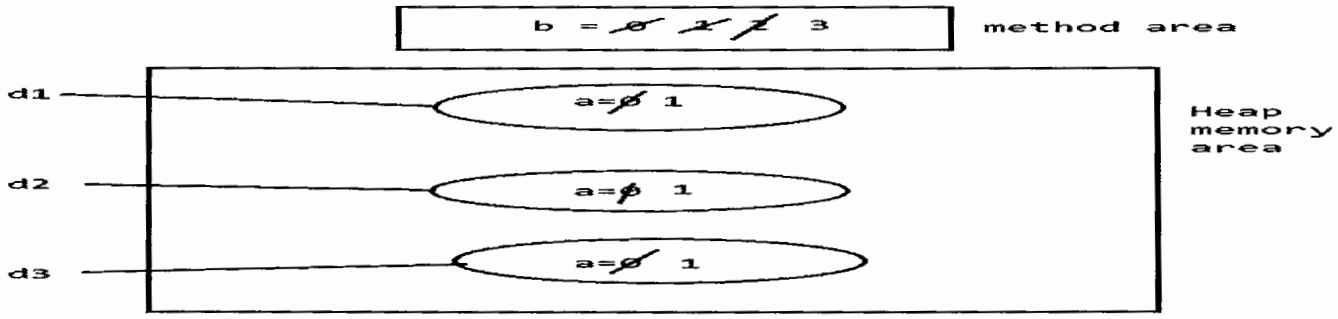
```
class StaticDemo3{
public  static void  main(String  args[]){
 Demo d = new Demo();
  d.display();
  d.show();
  }
}
```

3.when static variables are declared and not initialized with any value then static variables are automatically initialized with default values. If we want to initialize static variables with our own values then we can initialize satic variables only in 1 location 1. at the time of declaration of static variables

4.Same copy of static variables given to all objects of the particular class so these static variables can be sharable between all objects of the particular class

Eg:

```
class  Demo{
 int a;
static int b;
void   display(){
System.out.println("a="+a+"\t"+"b="+b);
}
}
class  StaticDemo4{
public static  void  main(String  args[]){
Demo d1 = new Demo();
 d1.display();//a=0     b=0
d1.a++;
d1.b++;
d1.display();//a=1     b=1
Demo d2 = new Demo();
d2.display();//a=0     b=1
d2.a++;
d2.b++;
d2.display();//a=1     b=2
Demo d3 = new Demo();
d3.display();//a=0     b=2
d3.a++;
d3.b++;
d3.display();//a=1     b=3
d1.display();//a=1     b=3
d2.display();//a=1     b=3
d3.display();//a=1     b=3
/*
d2.b--;
d1.display();//a=1     b=2
d2.display();//a=1     b=2
d3.display();//a=1     b=2
*/
 }
}
```



explain about main() method

-Java program execution will be begin from main() method

-JVM allways expect main() method like follows

```
public static void main(String args[]){
  //statements;
}
```

1. public:

public is keyword/ modifier / access specifier which indicates the particular entity can be accessed from any location. main() method should be declared as public so that JVM can execute the main() method from any location.

2. static:

static is keyword/ modifier which indicates the particular method can access directly by using class name. main() method should be declared as static so that JVM can access the main() method directly by using class name.

3. void:

void is keyword/ returntype which indicates the particular method does not return any value. JVM is not expecting any value to be return from main() method so that main() method should have the return type as void.

4. main():

main is the name of the method(identifier) which is fixed according to sun micro system's JVM implementation.

5. parameters of main() method (String args[])

parameters of main() method are mainly used to accept the command line arguments to resolve the problem of hard coding.

access modifiers(11)

access modifiers are java keywords which are always used along with other declaration and change the meaning of the that declaration.

Eg:

public, private,protected,static,abstract,final,.....

- sometimes we can also write multiple modifiers along with other declaration then order of modifiers can be anything.

Eg:

```
static public void main(String args[]){
}
public static void main(String args[]){
}
```

explain about System.out.println()

System.out.println() is statement used to display messages on the monitor.

System

System is a predefined class available in java.lang package

out

out is reference variable of PrintStream class which is holding an object of PrintStream class and declared in System class as static variable. so that we can access this out variable directly with the help of class name.

Println()

println() is the instance method of PrintStream class. If we want to access this method we must create an object for PrintStream class But we dont need to create any object for PrintStream class because the object for PrintStream class is already created in System class

Note

System.out is bydefault connected to monitor.

System.in is bydefualt connected to keyboard.

predefined code.

```
class PrintStream{
 public void println(xxx){
  }
 public void print(xxx){
 }
 public void printf(xxx){
  }
}
class  System{
  public  static final PrintStream  out= new PrintStream();
  public  static final InputStream  in= new InputStream();
}
```

local variables

-when we declare variables in side the method or constructor or any block then these variables are called local variables.

-local variables can be accessible only within the particular method or constructor or any block where they have been declared.

1.The memory allocated for local variables when method is invoked

2.The memory allocated for local variables in java stack of JVM

3.The memory allocated for local variables 1 time for every method invocation.

when ever the local variables are declared they will not initialize automatically with any default values like instance variables or static variables, It is the responsibility of the programmer to initialize the local variables and we can initialize local variables in following 2 locations

1.at the time of declaration of local variables (initialization level)

2.any where within same method before we use(assignment level)

```
//wap to demo on local variables
class  LVariableDemo{
 void show(){
   int  a=10;
   int b;
   int c;
  System.out.println(a);
   b=20;
  System.out.println(b);
  }
static  public  void   main(String  agrs[]){
  LVariableDemo  lv = new LVariableDemo();
  lv.show();
  }
}
```

Note:

1.if we declare any local variable and not using any where in the program then initializing local variable is optional otherwise initialization is mandatory.

2.parameters of any method are also considered as local variables

reference variables

-if any variable holds an object of any class or array then it is called as reference variable.

-a reference variable can be created inside the method or outside methods of the class as local, instance, static reference variable

//wap to demo on reference variables

```
class Demo{
   Demo d2 = new Demo(); //instance reference variable
static Demo d3 = new Demo(); // static reference variable
 public static void main(String args[]){
   Demo d1 = new Demo(); //local reference variable
   }
 }
```

what is the diff between primitive variables and reference variable

variables which are declared using primitive datatypes are called as primitive variables which can hold only primitive values like 10,10.2.'a',....

variables which are declared using any class or array are called as reference variables which can hold only object of any class or array.

Note:

we never store a null value into any primitive variable but we can store a null value into any reference variable.

int a = null; X -invalid Integer i =null; -valid

final keyword

-final is a keyword or modifier which can be applied for variables, methods and classes

-we use final keyword to maintain constant behavior in our application.

final variables

-generally the value of the variable can be changed any number of times. but if we want to declare the variables whose value is fixed an can not be changed then we have use " final " keyword.

-final variables are the constants whose value can not be changed at any time

-we can apply final keyword for instance variables, static variables, local variables, parameters reference variables,.....

final instance variables

if we declare instance variables using final keyword then we are only responsible for initializing the final instance variables and we can initialize final instance variables in following 2 locations.

1. at the time of declaration of final instance variables

2.inside the constructors

```
class  Demo{
  int a;
}
```

```
class  Demo{
  int a=10;
}
```

```
class Demo{
int a;
Demo(){
 a=10;
 }
 }
```

```
class Demo{
   int a;          //code is valid
void setValue(){   but here
   a=10;           instance variable
  }                is assigned
}
```

```
class  Demo{
 final  int a;
}
```

```
class  Demo{
 final   int a=10;
}
```

```
class Demo{
 final  int a;
 Demo(){
  a=10;
  }
 }
```

```
class Demo{
  final  int a;
 void setValue(){   //final instance
       a=10;          variables can not
   }                  be assigned
```

```
//wap to demo on final instance variables
    class  FIVariable{
    final  int a=10;
    final int b;
    final int c;
    final int d;
    final int e;

    FIVariable(){
    b=20;
    }
    void show(){
    System.out.println(a);
    b=20; //CTE: cannot assign a value to final variable b
    System.out.println(b);
    c=20; //CTE: cannot assign a value to final variable c
    System.out.println(c);
    System.out.println(d); //CTE:variable might not have been initialized
    }
    public  static void main(String  asr[]){
      FIVariable fiv = new FIVariable();
     fiv.show();
    }
   }
```

final static variables

if we declare static variables using final keyword then we are only responsible for initializing the final static variables and we can initialize final static variables in following 1 location

1. at the time of declaration of final static variables

```
class  Demo{
static int a;
}
```

```
class  Demo{
static  int a=10;
}
```

```
class Demo{
static  int  a;
Demo(){
a=10; //a is assigned but not
}       intialized
}
```

```
class Demo{
  static int a;
 void setValue(){
     a=10;// a is assigned but not
   }         intialized
  }
```

```
class  Demo{
static final int a;
}
```

```
class  Demo{
static  final  int a=10;
}
```

```
class Demo{
static  final  int  a;
Demo(){
a=10; // static variables
}     cant be intialized in
}     side the constructor
```

```
class Demo{
 static  final  int a;
    void setValue(){
        a=10;
      }
    }
```

```
// static variables
cant be intialized in
side the method
```

```
//wap to demo on final static variables
  class  FSVariable{
  final  static int a=10;
  final  static int b;
  final static int c;
  final static int d;
  FSVariable(){
  b=20; // CTE:cannot assign a value to final variable b
  }
  void show(){
  System.out.println(a);
  System.out.println(b);
  c=30;//CTE:cannot assign a value to final variable c
  System.out.println(c);
  }
  public  static void main(String  asr[]){
    FSVariable fsv = new FSVariable();
    fsv.show();
  }
  }
```

final local variables

if we declare local variables as final or non-final in both the cases we are only responsible for initializing the final local variables and we can initialize final local variables in following 2 locations

1.at the time of declaration of local variables (initialization level)

2.any where within same method before we use(assignment level)

```
class  Demo{
void  show(){
 int a;
}
}
```

```
class  Demo{
void show(){
 int a=10;
}
}
```

```
class Demo{
void show()
{
 int a;
a=10;
}
}
```

```
class  Demo{
void  show(){
 final int a;
}
}
```

```
class  Demo{
void show(){
final  int a=10;
}
}
```

```
class Demo{
void show()
{
final    int a;
a=10;
}
}
```

final parameters

parameters are the variables that we write within the method declaration to hold arguments or values that we pass while calling the method.

parameter values we can change inside the method if we require.

but if we don't want to change the values of parameters we have to declare parameters using final keyword.

```
class  FLVariable{
void show(){
final int a=10;
final int b;
final int c;
final int d;//CTE: variable d might not have been initialized
a=90; //CTE: cannot assign a value to final variable a
System.out.println(a);
b=20;
System.out.println(b);
System.out.println(d);
}
public  static void main(String  asr[]){
  FLVariable flv = new FLVariable();
  flv.show();
}
}
```

final reference variables

a reference variable refer to one object which can replaced by other objects any number of times. but if we don't want to replace the object of any reference variable with new object then we have to declare reference variables using final keyword.

```
class Demo{
int a;
int b;
Demo(int a,int b){
this.a=a;
this.b=b;
}
void show(){
System.out.println(a);
System.out.println(b);
}
}
class  FRVariable{
public  static void main(String  arg[]){
final Demo d = new Demo(10,20);
       // d = new Demo(100,200); //CTE: can not assign a
                                    final variable d
d.show();
d.a=1000;
d.b=2000;
d.show();
 }
}
```

Inheritance

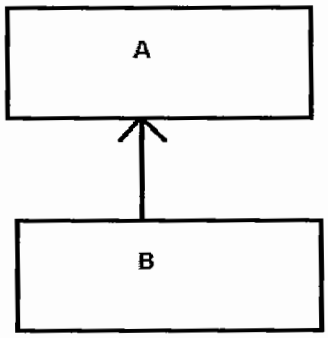
-inheritance is the one of most important concept of OOPS that we use in our projects
- inheritance means taking the properties of on class into another class
-in inheritance the class which is giving properties is called as parent class or super class or base class
-in inheritance the class which is taking properties is called as child class or sub class or derived class
-The main advantage of inheritance is re usability

types of inheritances

based on number of parent classes and child classes we have following 3 different types of inheritances in java.

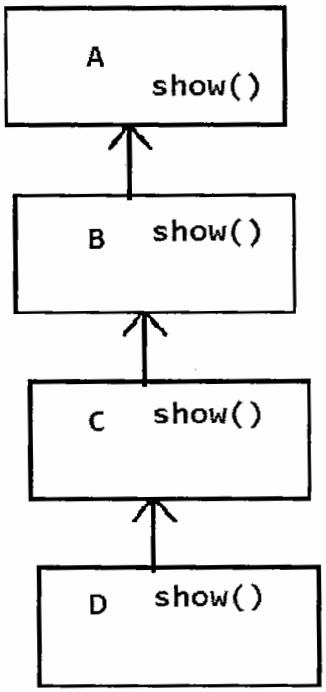
1. **single level inheritance**

single level inheritance means we contain 1 parent class and 1 child class.



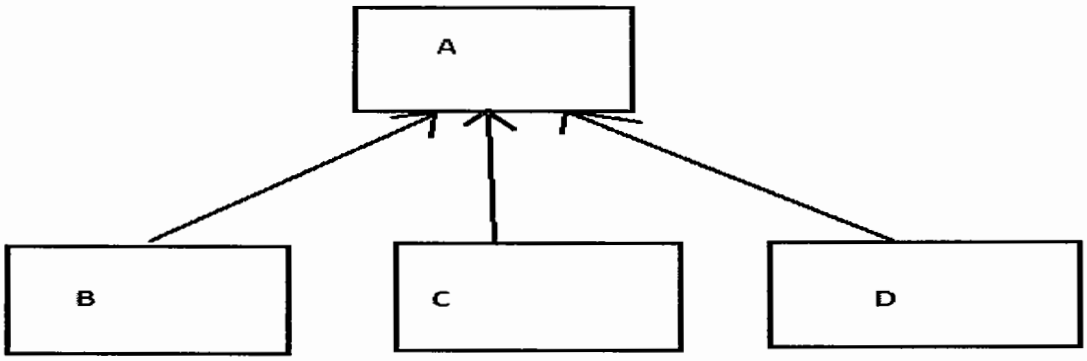
2. **multi level inheritance**

multi level inheritance means we contain 1 parent class and many child classes with intermediate classes.
intermediate classes means which sometimes work like parent class and some times like child classes.



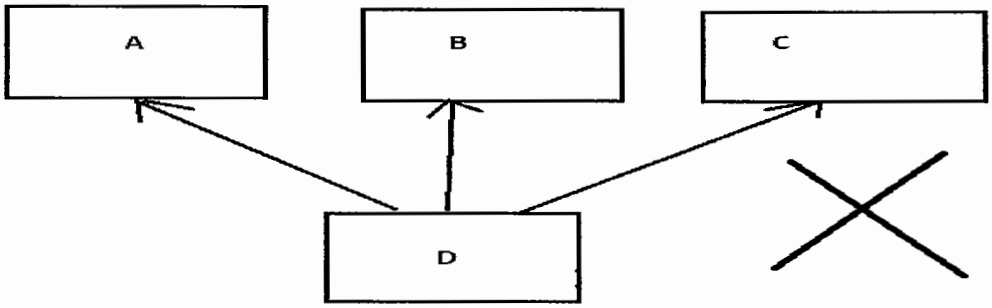
3. **hierarchical inheritance**

hierarchical inheritance means we contain 1 parent class and many child classes without any intermediate classes.



4. **multiple inheritance**

multiple inheritance means we contain multiple parent classes and one child classes. but this multiple inheritance is not supported in java.



? why java does not support multiple inheritance

In multiple inheritance child class is taking properties from multiple parent classes in this case if we have same property in all the paernt classes then it is a confusion to child class to decide to take the particular property from which parent class, hence multiple inheritance is not supported in Java.

Note:

In Java multiple inheritance is not supported using classes but supported using interfaces.

-In Java to achieve the inheritance we have to use a Java keyword called " extends ".

syntax:

```
class Class1{
    //members of the parent class
    }
class Class2 extends class1{
    //members of the child class
  }
```

```
//wap to demo on inheritance
class Parent{
int a = 10;
int b = 20;
void show(){
System.out.println("a="+a+"\tb="+b);
}
}
```

```
class Child  extends  Parent{
int c = 30;
int d = 40;
void display(){
a=300;
b=400;
System.out.println("a="+a+"\tb="+b);
System.out.println("c="+c+"\td="+d);
}
}
class  Inheritance1{
public  static void main(String args[]){
Parent p = new Parent();
p.show();
//p.display();-invalid
Child c = new Child();
c.show();
c.display();
c.show();
}
}
```

Note:

1.If we create an object for parent class then we can access only the members of Parent class

2.but if we create an object for child class then we can access the members of Both Parent class and Child class

Rules

1.In java we can create a class that extends only one class and we can not create a class that extends more than one class because java does not support multiple inheritance.

```
class A{
}
class B{
}
class C extends A,B{        X-invalid
}
```

2. In java cyclic inheritance is not supported.

```
class  Student extends Employee{        X-invalid    &   class Employee  extends  Employee{       X-invalid
  }                                                                                  }
  class Employee  extends  Student{
    }
```

3.In java every predefined class or user defined class are child classes of Object class either directly or indirectly so that all members of the Object class we can directly use in any class.

4.-when we declare a class and we are not extending from any class then compiler will create the class by extending from Object class automatically.

-but we declare a class by extending from any one class then compiler wont extend from Object class, hence we call Object class as java's super most class

```
class  A{                  ---->      class  A  extends java.lang.Object{
  }                                          }
class B extends A{         ---->   class  B  extends A{
      }                                            }
```

constructors in inheritance

-If we have constructor only in child class then when ever we create an object for child class then it will call child class constructor and executed.

-But if we have constructors in both parent class and child class and when ever we create an object for child class then first it will invoke child class constructor, child class constructor will invoke parent class constructor and this procedure will be continued up to java's super most class called Object class and next executed.

- Invoking of constructors is done from bottom to top i.e., from child class to parent class but execution is done from top to bottom i.e., from parent class to child class

```
//wap to demo on  constructors in inheritance
class Parent{
 Parent(){
    System.out.println("Parent class constructor");
  }
}
class Child1  extends  Parent{
 Child1(){
    System.out.println("Child1 class constructor");
  }
}
class Child2  extends  Child1{
 Child2(){
    System.out.println("Child2 class constructor");
  }
}
class  Inheritance2{
  public  static void main(String args[]){
   Child2  c = new Child2();
  }
}
```

o/p:

```
Parent class constructor
Child1 class constructor
Child2 class constructor
```

static variables and static methods in inheritance

If a Parent class contains static variables and static methods then we can access those members directly either by using Parent class name or Child class name.

Eg:

```
class Parent{
static int a = 10;
static int b = 20;
static void show(){
System.out.println("a="+a+"\tb="+b);
}}
class Child  extends  Parent{
static void display(){
a=300;
b=400;
System.out.println("a="+a+"\tb="+b);
}}
class  Inheritance3{
```

```
public  static void main(String args[]){
Child.show();
Child.display();
System.out.println("a="+Child.a+"\tb="+Child.b);
}
}
```

main() method in inheritance

we can also write main() method with in the parent class or child class.

Eg1:

Inheritance4.java

```
class Parent{
  public  static void main(String args[]){
    System.out.println("Parent class main() method");
   }
 }
class Child  extends  Parent{
  public  static void main(String args[]){
    System.out.println("Child class main() method");
   }
 }
```

compilation

javac Inheritance4.java

Execution

```
>java Child
Child class main() method
>java Parent
Parent class main() method
```

Eg2:

```
class Parent{
  public  static void main(String args[]){
    System.out.println("Parent class main() method");
   }
}
class Child  extends  Parent{
}
```

Execution

```
>java Parent
Parent class main() method
>java Child
Parent class main() method
```

Eg3:

```
class Parent{
}
class Child  extends  Parent{
  public  static void main(String args[]){
    System.out.println("Child class main() method");
   }
}
```

Execution

>java Child
Child class main() method
>java Parent
NoSuchMethodError: main

types of relation ships

if we want to access the information of one class inside another class then we can use any one of the following 2 relationships.

1. **Is-A relation**

-If class is extending from another class then it is called as Is-A relation.
-Here we can access class1 information inside the class2 directly without creating an object for class1.

Eg:

```
class Person{
 }
class Employee extends Person{
}
class Student extends Person{
}
```

2. **Has-A relation**

-If a class contains other class object then it is called as Has-A relation.
-Here we can access class1 information inside the class2 only by using object of class1.

Eg:

```
class Heart{
  }
class Person{
 Heart h = new Heart();
 }
```

polymorphism

-Polymorphism means defining one entity with multiple bahaviors
-The main Advantage of Polymorphism is flexibility, because using one entity we can perform multiple operations

Eg:

student, pavan kalyan, println(),...

In java we have following 2 types of poly morphisms.

1. **compile time poly morphism**

- If polymorphism is decided at the time of compilation by java compiler then it is called as compile time polymorphism
- by using the concept of method overloading we can achieve this compile time poly morphism.

2. **runtime poly morphism**

- If polymorphism is decided at the the time of execution time by JVM then it is called as run time polymorphism.
- by using the concept of method overriding we can achieve this runtime poly morphism.

method overloading

method overloading means declaring multiple methods with same method name but having different method signature. In method overloading while writing the method signature we have to follow following 3 Rules

1.method name must be same for all methods
2.list of parameters must be different like different type of parameters, different number of parameters, different order of parameters.
3.returntype is not considered in method overloading, it means we never decide method overloading with returntype

```
void show(int x,int y);
void show(double x,double y);        // methods are overloaded based on type of parameters

void show(int x);
void show(int x,int y);              // methods are overloaded based on number of parameters

void show(int x,double y);
void show(double x,int y);           // methods are overloaded based on order of parameters

void show(int x,int y);
void show(int y,int x);              // methods are not overloaded because based on parameter name we can not overload

void show(int x,int y);
int show(int x,int y);               // methods are not overloaded because based on return type we can not overload

void show(int x,int y);
int show(double x,double y);         // methods are overloaded based on type of parameter but not based on retur type

void show(int x,int y);
void add(double x,double y);         // methods are not overloaded because 2 method names are different
```

Note:

in method overloading the method invocation will be linked or bind with appropriate method definition at the time compilation hence it is called as compiletime polymorphism

once the linking or binding is done that we never modify at run time so it is called as static binding

here linking or binding is done while compilation only so it is called as early binding.

//wap to demo on method overloading
(using different types of parameters)

```
class Demo{
 void show(int a){
   System.out.println("int parameter a="+a);
 }
 void show(double a){
   System.out.println("double parameter a="+a);
 }
 void show(String a){
   System.out.println("String parameter a="+a);
 }
 void show(boolean a){
  System.out.println("boolean parameter a="+a);
 }
}
```

```
class Polymorphism1{
public static  void main(String ars[]){
Demo  d = new Demo();
d.show(20.3);
d.show(true);
d.show("abc");
d.show(10f);
d.show(10);
d.show(new String("abc"));
d.show('A');
}
}
```

Rules

1. we can overload only instance methods or only static methods or both at a time.

```
//method overloading with different number of parameters
class Demo{
 static void show(int a){
   System.out.println("method with 1 parameters");
  }
static  void show(int a,int b){
   System.out.println("method with 2 parameters");
  }
 void show(int a,int b,double c){
   System.out.println("method with 3 parameters");
  }
  void show(){
   System.out.println("method with 0 parameters");
  }
}
class Polymorphism2{
public static  void main(String ars[]){
Demo  d = new Demo();
d.show();
d.show(10);
d.show(10,20,30.2);
d.show(10,20);
}
}
```

2.we can overload methods with in the same class and we can also overload methods with in 2 different classes which are having IS-A relation.

```
//method overloading with different order of parameters
class A{
void show(boolean a,int b){
   System.out.println("method with boolean,int parameters");
 }}
class B extends A{
 void show(int a,boolean c){
   System.out.println("method with int,boolean parameters");
 }}
```

```
class Polymorphism3{
public static  void main(String ars[]){
B b= new B();
b.show(10,true);
b.show(true,10);
}
}
```

3. we can also overload main() method but JVM always invokes only main() method which takes String[] as parameter, if we want to access other oveloaded main() methods then we have to call explicitly

```
class Polymorphism4{
public static  void main(String ars[]){
System.out.println("main() method with String[] as parameters");
Polymorphism4.main(true);
Polymorphism4.main(90);
int iarr[] = {45,45};
Polymorphism4.main(iarr);
double darr[]= {10.0,11.0};
Polymorphism4.main(darr);
}
public static  void main(int  ars[]){
System.out.println("main() method with int[] as parameters");
}
public static  void main(double ars[]){
System.out.println("main() method with double[] as parameters");
}
public static  void main(int  a){
System.out.println("main()method with int as parameters");
}
public static  void main(boolean a){
System.out.println("main()method with boolean as parameters");
}
}
```

? why should we go for method overloading

when we want to maintain the flexibility in our application like using one method performing sveral operations then we can use this method overloading.

method overriding

- If we want to achieve the run time polymorphism then we have to use method overriding.
- Method overriding means declaring 2 methods with same method signature in 2 different classes which are having IS-A relation.
- While Method overriding and writing the method signature we must follow following rules.
    1. method name must be same
    2. list of parameters must be same
    3. return type must be same

```
// wap to demo on method overriding
class Parent{
 void show(int a){
   System.out.println("Parent class show() method a="+a);
  }}
class Child extends Parent{
```

```
int x=10;
/*overriding the show() method*/
 void show(int a){
   System.out.println("Child class show() method a="+a);
  }
/*display() method which is not overriden*/
 void display(int a){
   System.out.println("Child class show() method a="+a);
  }
}
class Polymorphism5{
public static  void main(String ars[]){
Parent p;
p = new Parent();
p.show(10); //Parent class show() method a=10
p = new Child();
p.show(10);//Child class show() method a=10
//p.display(10); // invalid
//System.out.println(p.x) //invalid
}}
```

-In method overriding the method invocation will be linked with either child class method definition or parent class method definition based on the object that reference variable contains, It means that objects are available at run time and this kind of linking or binding can be decided at run time hence it is called as run time polymorphism or dynamic polymorphism.

-This linking is decided at run time after the object is created so it is called as late binding.

Note:

-A class reference variable can hold its object or all its child class objects.

```
Object o1 = new String();
Object o2 = new Integer();
Object o3 = new Student();
String s = new Integer();     X- invalid    (Here String and Itneger classes are not having IS-A relation)
```

-if parent class reference variable holds child class object then we can access every member of Parent class and we can access only overridden mehods of Child class, but we can not access original methods and variables of Child class to resolve this problem we have to convert the child object from parent reference variable to child class reference variable(down casting)

?Why should we go for Method overriding

when ever child class don't want to use definition written by the Parent class and want to use its own logic then we have to use method overriding it means we have to override the same method with new definition inside the child class.

Rules

1.In Method overriding we must declare 2 methods in 2 different classes which are having Is-A relation
But we never do Method overriding with in the same class

2. we can override 2 instance methods (method overriding), we can override 2 static methods (method overhiding) but we cant override 1 instance method and 1 static method

```
//wap to demo on method overriding
class Parent{
static  void show(){
System.out.println("Parent Class show() method");
}}
class Child  extends Parent {
```

```
static void show(){
System.out.println("Child Class show() method");
}
}
class  Polymorphism6{
public  static  void  main(String  args[]){
Parent p1;
p1 = new Child();
p1.show();  //Parent Class show() method
p1 = new Parent();
p1.show(); //Parent Class show() method
p1 = null;
p1.show();  //Parent Class show() method
Child  c = new Child();
 c.show();  //Child Class show() method
}
}
```

constructor overloading

-Constructor overloading means declaring multiple constructors with different method signature.

-Main advantage of constructor overloading is flexibility like providing different ways to create an object for a class.

```
//wap to demo on constructor overloading
class  Cube{
int l,b,h;
Cube(){
l=b=h=5;
}
Cube(int x){
l=b=h=x;
}
Cube(int l,int b,int h){
 this.l=l;
 this.b=b;
 this.h=h;
}
void area(){
 System.out.println("Area of Cuboid: "+(l*b*h));
}
}
class Polymorphism7{
public  static  void main(String args[]){
 Cube  c1  = new Cube();
   c1.area();
 Cube  c2  = new Cube(7);
  c2.area();
  Cube  c3 = new Cube(2,5,4);
   c3.area();
 }
}
```

Note:

1.constructor overloading must be done with in the same class

2.when we write parameter constructor then it is recommended to write 0 parametrized constructor as well.

final keyword

final is a keyword or modifier which can be used on variables , methods, classes.

final variables

If variables are declared using final keyword then the value of that variable can not be changed at any time which are called as constants. we can apply final keyword for instance variables, static variables, local variables.....

final methods

-if any method is declared then the definition of the method we can change by overriding the method in child classes

-but if we want to declare the method whose definition can not be changed by overriding the method in child classes then the particular method must be declared as final methods.

-a final method is a method which can not be override the method in child classes we can also call this final methods as must not overridable methods.

Eg:

```
class Parent{
final void show(){
   System.out.println("Parent Class show() method");
  }
}
class  Child extends Parent{
/*
//final method can not be ovveriden
void show(){
   System.out.println("Child Class show() method");
  }*/
}
class  FinalMethodDemo{
  public  static void main(String args[]){
 Child c = new Child();
  c.show();
   }
}
```

Note:

1. we can not override a final method but we can call and access it

2. final methods can be overloaded

final classes

-if any class is declared any body can extend the particular class but we don't want to allow the others to extend our class then we should declare our class as final.

-a final class is a class which can not be extend by any other class by declaring final class we can restrict others to use only Has-A relation because here Is-A relation is not possible

- we can consider all the methods of final class as final methods because which can not be override, But all the variables of final class are not final.

Eg:

```
final class Parent {
int a=10;
int b=20;
 void show(){
```

```
   System.out.println("a="+a);
   System.out.println("b="+b);
  }
}
/*
//we can not hinherit from final class Parent
class  Child extends Parent{
}
*/
class  FinalClassDemo{
  public  static void main(String args[]){
    Parent  p = new Parent();
     p.show();
  p.a=100;
  p.b=200;
  p.show();
 }
}
```

Note:

1. A final class can not be inherited by any other class but a final class can extend any other class and also override the methods if needed.
2. we can create an object for a final class and we can access the members of final class.

this keyword

-this keyword isreference and holds current class instance/object so that we can access all the members of current class using this this keyword with in the same class.

-writing this keyword is sometime optional sometimes mandatory.

-if there is no confusion between local variables and instance variables then writing this keyword is optional. In this case compiler will write this keyword and refer to instance variable.

-but if there is a confusion between local variables and instance variables then writing this keyword is mandatory. In this case compiler will not write this keyword and refer to local variable. But if we want to refer to instance variable we have to write this keyword explicitly.

```
//wap to demo on this keyword
class Demo{
int a=10;
int b=20;
static int x=99;
 void show(){
    int a=30;
    int c=40;
    System.out.println(this.a);//this keyword is mandatory
    System.out.println(a);//local variable
    System.out.println(this.b);//this keyword is optional
    System.out.println(c);//local variable
    System.out.println(this.x);//this keyword is optional
    this.display();//this keyword is optional
    this.printing();//this keyword is optional
  }
 void display(){
  System.out.println("this is display() method");
```

```
  }
 static void printing(){
  System.out.println("this is printing method");
  }
}
class  ThisDemo1{
  public  static void main(String args[]){
  Demo  d = new Demo();
   d.show();
  }
}
```

Note:

-with this keyword we can access both instance and static members

-we can use this keyword inside the constructor,inside the instance methods of the class but we can not use this keyword inside the static methods of the class

this():

this() is used to invoke and access the 0 parametrized constructor of the same class inside another constructor.

this(...)

this(...) is used to invoke and access the parameterized constructor of the same class inside another constructor.

```
//program to demo on this(), this(...)
class  Demo{
Demo(){
this(10);
System.out.println("this is 0 parameterized constructor");
}
Demo(int x){
System.out.println("this is 1 parameterized constructor");
}
Demo(int x,int y){
this();
System.out.println("this is 2 parameterized constructor");
}
}
class  ThisDemo2{
public  static  void  main(String  args[]){
Demo d = new Demo(10,20);
}
}
```

o/p:

this is 1 parameterized constructor
this is 0 parameterized constructor
this is 2 parameterized constructor

Rules

1.calling this() or this(...) must be the first statements in the constructor.
2. calling this() or this(...) must not be in side the regular method
3. we can call this() or this(...) at most 1 time
4. calling this() or this(...) must not provide a recursive constructor invocation

super keyword

-super keyword refer to parent class instance or object so that we can access instance and static members of the Parent

class with in the child class
-writing super keyword is some times mandatory ,sometimes optional
-if there is no confusion between instance variables of Parent class and instance variables of Child class then writing super keyword is optional. But In this case compiler will write this keyword and refer to instance variable of Child class.
-but if there is a confusion between instance variables of Parent class and instance variables of Child class then writing super keyword is mandatory. Even In this case compiler will write this keyword only and refer to instance variable of Child class. But if we want to refer to instance variable of Parent class we have to write superkeyword

```
//wap to demo on super keyword
class Parent{
int a=10;
int b=20;
void show1(){
System.out.println("Parent class show1() method");
}
void show2(){
System.out.println("Parent class show2() method");
}}
class  Child extends Parent{
int a=30;
int c=40;
void show1(){
System.out.println("Child  class show1() method");
}
void show3(){
System.out.println("Child class show3() method");
}
void show4(){
int a=50;
int d=60;
System.out.println(super.a);//super is mandatory(Parent class a val)
System.out.println(this.a);//this is mandatory(Child class a value)
System.out.println(a);//local variable a value
System.out.println(b);//super or this is optional(Parent class b value)
System.out.println(c); //this is optional (Child class c value)
System.out.println(d); // local variable d value
super.show1(); //super is mandatory (calls Parent class show1())
this.show1();// this is optional(calls Child class show1())
show2(); // super or this optional (calls Parent class show2())
this.show3(); // this optional (calls Child class show3())
}}
class SuperDemo1{
public  static  void main(String args[]){
  Child c= new Child();
   c.show4();}}
```

Note:

1.a super keyword can access both instance and static members of the Parent class with in child class instance methods, but we can not use super keyword within any static method of child class.

2.we don't write any this keyword or super keyword compiler always write this keyword only but not any super keyword

3. super keyword in child class always point to the variables or methods of its immediate parent class.

Eg:

```
class Cone{
int a=10;
}
class Ctwo extends  Cone{
int a=20;
}
class Cthree extends  Ctwo{
int a=30;
void show(){
System.out.println(super.a); //20
Cone c = new Cone();
System.out.println(c.a);
}
}
class SuperDemo2{
public static  void main(String ars[]){
Cthree  c = new Cthree();
 c.show();
}
}
```

super()

this super() is used to call the parent class 0 parametrized constructor in side the constructors of Child class.

super(...)

this super(...) is used to call the parent class parameterized contructor in side the constructors of Child class.

```
//wap to demo on super() or super(...)
class Parent{
Parent(){
System.out.println("Parent  class 0 parameterized constructor");
}
Parent(int x){
this();
System.out.println("Parent  class 1 parameterized constructor");
}
}
class Child extends Parent{
Child(){
this(10,20);
System.out.println("Child  class 0 parameterized constructor");
}
Child(int x){
this();
System.out.println("Child  class 1 parameterized constructor");
}
Child(int x,int y){
super(10);
System.out.println("Child  class 2 parameterized constructor");
}
}
```

```
class SuperDemo3{
 public static  void main(String ars[]){
  Child c = new Child(10);
 }
}
```

Note:

inside the constructor if we don't write either this() or super() then compiler will automatically writes super() always. so that every child class constructor will invoke its parent class constructor by default.

Rules

1.calling super() or super(...) or this() or this(...) must be the first statement in the constructor
2.calling super() or super(...) or this() or this(...) must not be in side the regular method
3.we can call super() or super(...) or this() or this(...) at most 1 time
4.calling super() or super(...) or this() or this(...) must not provide a recursive constructor invocation

ellipsis (...) operator

-ellipsis (...) operator introduced in java 1.5 version which is used to create method with variable number of arguments
-ellipsis operator internally maintained just like an array.
-ellipsis operator used in only 1 location in entire java that is while writing the
 parameters of the method.

```
// wap to demo on ellipsis (...)  operator
class Demo{
void  sum(int... a){
//System.out.println("No.of Parameters="+a.length);
int c=0;
for(int i=0;i<a.length;i++){
c  = c + a[i];
}
System.out.println("sum="+c);
}
public static void main(String args[]){
 Demo d = new Demo();
  d.sum(10);
  d.sum(10,20);
  d.sum(10,20,30);
  d.sum(10,20,30,450);
  d.sum(10,13,60,70,70,60,20);
  d.sum(new int[]{21,34,56});//ellipsis operatator takes array
  d.sum(new int[]{21,34,56,67,78,78,89});
 }
}
```

Note:

-In a method/constructor we can also write normal parameters along with parameters which are declared using ellipsis operator but in this case first we have to write normal parameters and followed by parameters which are declared using ellipsis operator

```
  void sum(int... ar,int x){                X-invalid
          }
  void sum(int x,int... ar){                -valid
          }
```

-we can also write our main() method using ellipsis (...) operator

```
  public static void main(String... args){
```

```
    }
```

? what is the diff between (int... args) and (int[] args)

in a method if we use ellipsis (...) operator we can pass any number of arguments but if we use array as parameter then we have to pass only 1 argument that is an array which may contain any number of values.

Eg:

```
class Demo{
void add(int... a){
int c=0;
for(int i=0;i<a.length;i++){
c = c + a[i];
}
System.out.println("sum="+c);
}
void mul(int[] a){
int c=1;
for(int i=0;i<a.length;i++){
c = c * a[i];
}
System.out.println("mul="+c);
}
public static void main(String args[]){
 Demo d = new Demo();
  d.add(10);
  d.add(10,20,30);
  d.add(new int[]{10,20,30,40});
  d.mul(new int[]{10,20,30});
  d.mul(new int[]{10,20,30,40,50,40});
 //d.mul(10,20,30); -invalid because mul() take only 1 argument
 }
}
```

Note:

- we can not overload 2 methods which takes array as parameter and second one takes ellipsis operator as parameter.

```
void add(int... a){                X-invalid
}
void add(int[] a){
}
```

instance block

if we write a group of characters in between a pair of curly brackets {} inside the class and outside the methods without any static word then it is called as instance blocks

syntax:

```
 {
  //group of statments
 }
```

-instance blocks are mainly used to initialize the instance variables

-when instance variables are declared they automatically initialized with default values but if we want to initialize instance variables with our own values then we can initialize in following locations.

1. at the time of declaration 2. inside the contructor 3. inside the instance blocks

- we can write any number of instance blocks and any where in the class outside the methods.

procedure how jvm execute the instance blocks

-when ever an object is created then all the instance blocks will be executed top to bottom and next corresponding Constructor will be executed.
- all instance blocks are executed top to bottom for every object 1 time.

static blocks

if we write a group of characters in between a pair of curly brackets {} inside the class and outside the methods with static keyword then it is called as staic blocks

syntax:

```
static {
  //group of  statements
 }
```

- static blocks are mainly used to initialize the static variables
- when static variables are declared they automatically initialized with default values but if we want to initialize instance variables with our own values then we can initialize in following 2 locations.

1. at the time of declaration 2. inside the static blocks

- we can write n number of static blocks any where in the class outside the methods.

procedure how jvm execute the stactic blocks

-at the time of class loading all the static blocks will be executed top to bottom and next jvm will search for main() method and start executing main() method.
-static blocks will be executed top to bottom only for 1 time.

//wap to demo on instance blocks and static blocks

```
class  BlocksDemo{
    {
    System.out.println("instance block1"); (4) (8) (12)
    }
   static {
   System.out.println("static  block1"); (1)
   }
   BlocksDemo(){
   System.out.println("0 parameterized constructor"); (6)  (10)
   }
   public static void main(String  args[]){
    BlocksDemo  o1 = new BlocksDemo(); (3)
    BlocksDemo  o1 = new BlocksDemo();  (7)
    BlocksDemo  o3 = new BlocksDemo(10);  (11)
    }
   {
   System.out.println("instance block2"); (5) (9) (13)
   }
   static {
   System.out.println("static  block2");  (2)
   }
   BlocksDemo(int a){
   System.out.println("parameterized constructor");  (14)
    }
  }
```

//writing a program without main() method

```
class  BlocksDemo1{
static{
 int a,b,c;
 a=10;
 b=20;
```

```
c=a+b;
 System.out.println("Addition="+c);
}
}
```

type casting

-type casting is the process of converting value from one type to another type.
-we always do typecasting between 2 different data types which are compatible.
-in between 2 same data types typecasting is not required.
-in java we have following 2 types of type castings
1.type casting wrt primitive data types 2.type casting wrt reference types

1. Type casting wrt primitive data types

-Type casting wrt primitive data types means converting value from one primitive data type to other primitive data type
-Type casting can be done only between compatible data types
-compatible data types are byte,short,int,long,float,double,char
-we have following 2 types of Type casting wrt primitive data types

1. widening

widening means converting the value from lower data type into higher data type.

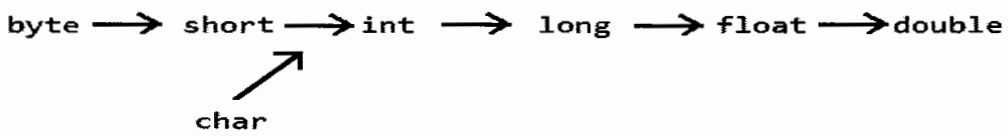
syntax:

higherdatatype = (higherdatatype) lowerdatatype ;

-here data type specified in between the pair of parenthesis is called type casting.
-in widening writing the type casting is optional
-if we dont write any type casting then compiler will write the type casting automatically hence it is called as implicit type casting.

```
//wap to demo on widening
class TypeCasting1{
public static void main(String args[]){
 int a=10;
 double b;
 b=(double)a; //typecasting is optional
 System.out.println("b="+b);
 byte p=90;
 int q;
 q=(int) p;//typecasting is optional
 System.out.println("q="+q);
}}
```

automatic typecasting/ implicit type casting



2. narrowing

narrowing means converting the higher data type value into smaller data type.

syntax:

lowerdatatype = (lowerdatatype) higherdatatype;

-in narrowing writing the type casting is mandatory
-in narrowing if we dont write any type casting then compiler wont write any typecasting because there is a chance of

loss of some data so that user has to write the typecasting explicitly hence it is called as explicit type casting.

```
//wap to demo on narrowing
class TypeCasting2{
public static void main(String args[]){
 double a=10;
 int b;
 b=(int) a; // typecasting is mandatory
 System.out.println("b="+b);
 int p=128;
}
}
```

Note:

we can not do typecasting between 2 incompatible data types for example we can not do type casting between int and boolean types.

Eg:

```
int a=10;                          (or)    Int a=10;
boolean b;                                 boolean b;
b=a; //CTE: incompatible types             b=(boolean) a; //CTE: inconvertible types
```

2. type casting wrt reference types

-Type casting wrt reference types means converting the object from one reference type to another reference type

-But Type casting wrt references can be done only between compatible types

-The two references said to be compatible if and only if its corresponding classes having Is-A relation

-Type casting wrt references is also classified into following 2 types

1. upcasting

up casting means storing the child class object into the parent class reference.

syntax:

parentreferencetype = (parentreferencetype) childreferencetype

- in upcasting writing typcasting is optional

2. downcasting

downcasting means storing the Parent class object into the child class reference.

syntax:

childreferencetype = (childreferencetype) parentreferencetype

- in downcasting writing typcasting is mandatory

```
//wap to demo on upcasting and downcasting
class Parent{
 void show(){
  System.out.println("This is Parent class Method");
 }}
class Child extends Parent{
void show(){
System.out.println("This is Child class Method");
}
}
class TypeCasting3{
public static void main(String args[]){
Parent p;
p = new Parent();
p.show();
p = (Parent) new Child(); //typecasting optional (upcasting)
```

```
p.show();
Child  c =(Child)  p; //typecasting is mandatory ( downcasting)
c.show();
/*
 //RTE:ClassCastException:Parent cannot be cast to Child
 p = new Parent();
 Child  c1 = (Child) p; //downcasting
 c1.show();   */
}
}
```

Note:

In downcasting while converting Parent class reference type into child class reference type then Parent class reference should contain child class object. otherwise we get a runtime exception saying ClassCastException: Parent cannot be cast to Child

abstract methods

-when we declare a method we must provide method body or method definition for particular method,but if we want to declare any method without any body then that method must be declared using a java keyword called abstract.

- abstract method is a method which does not contain any body whose definition is provided in child class of corresponding class.
- we can call abstract methods as must overridable methods

syntax:

```
abstract  returntype  methodname(parameters);
```

Note:

abstract methods can not have any body and must be terminated by ;

```
abstract void show(){       //invalid
   }
abstract void show();       //valid  (abstract method)
      void show();          //invalid
     void show(){           //valid (concrete method)
          }
```

abstract classes

-if a class contains at least one abstract method we must declare class using a java keyword called abstract.

-an abstract class is a class which contains 0 or more abstract methods.

-for abstract class we can not create an object, but we can create a reference

-but if we want to create an object for abstract class we have to create the child class for abstract class, then by taking the help of this child class we can create an object for abstract class.

- we can consider abstract class as must inheritable class.

syntax:

```
abstract  class  classname{
   //members
   0 or more astract methods
 }
```

```
//wap to demo on abstract classes and abstract methods
abstract  class  Shape{
  int l;
  int b;
 Shape(int l,int b){
    this.l=l;
    this.b=b;
```

```
  }
 abstract void area();
}
class Square extends Shape{
 Square(int side){
 super(side,side);
  }
 void area(){
    System.out.println("Area of Square="+l*b);
   }
}
class Rectangle extends Shape{
Rectangle(int l,int b){
   super(l,b);
  }
 void area(){
    System.out.println("Area of Rectangle="+l*b);
  }
}
class AbstractDemo{
public static void main(String args[]){
/*
 // CTE: Shape is abstract can not be instantiated
 // it means we can not create an object for abstract class directly
   Shape s= new Shape(); */
 Shape s; //creating reference for abstract class (valid)
  s = new Square(5);
  s.area();
  s = new Rectangle(3,4);
  s.area();
  }
}
```

concrete methods

if a method is declared with method body then it is called as concrete method otherwise if it is not containg any method body then it must be declared using abstract keyword and it is called as abstract method.

concrete classes

if a class declared without any abstract methods then it is called as concrete class otherwise if a class contains atleast one method as abstract method then particular class must be declared using abstract keyword and it is called as abstract class

-a concrete class contains instance variables,static variables, constants, concrete methods, etc except abstract methods

-but a abstract class contains instance variables,static variables, constants, concrete methods, etc including abstract methods

IQ: can we write main() method in abstract classes

yes we can write main() method inside abstract classes

Demo.java

```
abstract class Demo{
public static void main(string args[]){
 System.out.println("main() method");
  }
```

```
}
```
Cimpilation:
> javac Demo.java
Execution:
> java Demo
o/p:
main() method

Note:

1. final keyword can be applied for variables, methods, classes
2. static keyword can be applied for variables, methods, inner classes but not for outer classes
3. abstract keyword can be applied for methods, classes but not for any level of variables.

```
abstract int a,b;   X - invalid
```

illegal combination of modifiers for a method

-we never delcare a method as final and abstract because final keyword says no overrding for method where abstract keyword says method must be override.

```
abstract final void area(); X -invalid
```

- we never delcare a method as static and abstract because static methods can be accessed directly by using the class name and will expect method definition where abstract keyword says no method body for the particular method

```
abstract static void area(); X -invalid
```

illegal combination of modifiers for a class

-we never delcare a class as final and abstract because final keyword says class must not inheritable class but where abstract keyword says class is must inheritable class

```
abstract final class Demo{    X -invalid
}
```

Interfaces

- an abstract class can contain both concrete methods and abstract methods or some times it may contain no abstract method, so that using abstract classes as a reusable components will be a problem. to maintain the pure reusable components then we take the help of interfaces.
- an interface is one kind of class which contains only a group of abstract methods which can be used as a reusable component.
- an interface can be considered as a business agreement.
- to create the interfaces we use java keyword called "interface"

syntax:

```
interface  InterfaceName{
 //members;
 }
```

Eg:

```
interface Calc{
 int a=12;
 void add();
 void sub();
}
```

Note:

- when ever any java program is compiled then java compiler will generate .class file for every class available in the program and also create the .class file for every interface available in the program.

-If we compile the above program then compiler will generate .class file like Calc.class.

Displaying the profile of Calc.class

```
> javap Calc
interface Calc{
  public static final int a;
  public abstract void add();
  public abstract void sub();
}
```

Note:

-All the variables declared in interface are public static final by default whether we specify or not

-All the methods declared in interface are public abstract by default whether we specify or not

public:

all the variables and methods declared in interface are public so that they can be accessible from any where.

static:

variables declared in itnerface are bydefault static so that they can be accessible directly by using the interface name.

final:

the variables declared in interface are bydefault final it means they are constant whose value can not be changed.

abstract:

all the methods declared in interface are abstract because they dont contain any method body.

Note:

1. For an interface we can not create any object directly
2. but we can create a reference for interface

- if we want to create an object for any interface then we must create a class that inherit the particular interface and provide the implementation for all the abstract methods of that interface.

-the members of interface can be inherited into any class by using a keyword called " implements " this concept is also considered as IS-A relation.

syntax:

```
class classname implements interfacename{
  // members
 }
```

Rules

1.-Once an interface is implemented by any class then that class must provide the implementation for all the methods available in the particular interface.This class is also called as implementation class or child class.

2.-For example if our class is not providing implementation for at least 1 method then our class must be declared as abstract

3.-We can not create an object for abstract class or interface but we can create an object only for implementation class.

-once an interface is created then any number of classes can inherit that interface

```
//program to demo on interfaces
interface Vehicle{
 void travelling();
}
class Bus implements Vehicle{
public void travelling(){
 System.out.println("Travelling on the dividers");
}
}
class TrackBus implements Vehicle{
public void travelling(){
 System.out.println("Travelling on the roads");
}}
```

```
class  AirBus  implements Vehicle{
public void travelling(){
 System.out.println("Travelling on the rivers");
}
}
class Interface1{
public  static void main(String args[]){
   Bus  b = new Bus();
    b.travelling();
   TrackBus  tb = new TrackBus();
    tb.travelling();
   AirBus  ab = new AirBus();
    ab.travelling();
  }
}
```

-variables in interface are final so while declaring the variable in interface specifying the initial value is mandatory

-variables in interface are public static so that we can access directly in side the implementation classes but if we want to use in any other class we can access by using InterfaceName directly.

Eg:

```
interface Shape{
 double  PI=3.14;
 void   area();
}
class  Circle  implements  Shape{
int r;
Circle(int r){
this.r=r;
}
public  void area(){
 System.out.println("Area of Circle:"+PI*r*r);
}}
class  Sphere  implements  Shape{
int r;
Sphere(int r){
this.r=r;
}
public  void area(){
 System.out.println("Area of Sphere:"+4/3.0*PI*r*r*r);
}
}
class  Interface2{
 public  static void main(String args[]){
 Shape s;
 s = new Circle(5);
  s.area();
s = new Sphere(4);
s.area();
 System.out.println(Shape.PI);
}}
```

-multiple inheritance is not possible in java using classes because in java a class can not extend more than 1 class.
- but we can achieve this multiple inheritance by taking the help of interfaces because in java a class can implement any number of interfaces.

```
interface Calc1{
   void add(int x,int y);
   void sub(int x,int y);
   void mul(int x,int y);
}
interface Calc2{
   void sub(int x,int y);
}
interface Calc3{
   void mul(int x,int y);
}
interface Calc4{
   void div(int x,int y);
}
class Demo{
}
class Calculator implements Calc1,Calc2,Calc3,Calc4 {
public void add(int a,int b){
 System.out.println("Addition: "+(a+b));
}
public void sub(int a,int b){
 System.out.println("Subtraction: "+(a-b));
}
public void mul(int a,int b){
 System.out.println("Multiplication: "+(a*b));
}
public void div(int a,int b){
 System.out.println("Division: "+(a/b));
}
}
class Interface3{
public static void main(String args[]){
 Calc1 c = new Calculator();
  c.add(10,20);
  c.sub(10,2);
 }
}
```

Rules

```
interface  A{
}
inteface B{
}
inrerface C{
}
```

```
class D{
}

class E{
}
```

```
class  F extends D{ ✓  // a class can extend other class
}

class  G extends A{ ✗ //a class can not extend other interface
}

class J extends  D,E{ ✗ // a class can not extend more than one class
}

class  H imeplements A,B,C{ ✓ // a class can implement any number of
}                                interfaces

class HH implements D{ ✗  //a class can not implement other class
}

interface I extends A,B,C{ ✓ // an interface can extend any number of
}                               other itnefaces

inteface K implements A{ ✗ //an interface can not implement other interface
}

interface KK implements D{ ✗ //an interface can not implement a class
}

inteface L extends D{ ✗ // an interface can not extend other classes
}

class M extends D implements A{ ✓ //a class can extend a class and can
}                                   implement an interface

class N extends D implements A,B,C{ ✓ //a class can extend a class and can
}                                       implement any number of interfaces

class O  implements A,B,C extends D{ ✗ a class can extend a class and can
}                                      implement any number of interfaces
                                       at a time but first we have to write
                                       extends keyword followed by
                                       implements keyword
```

abstract classes	interfaces
1.abstra class contains both abstract methods concrete methods	1. but interface contains only abstract methods
2.writing abstract keyword mandatory for abstract methods	2. writing abstract keyword optional
3. It contains both public & non public members	3. It contains only public members
4. It contains both static & non-static methods	4.It contains only instance(nonstatic)methods
5.It contains both static & non-static variables	5. It contains only static variables
6.It contains both final &non-final variables	6. It contains only final variables
7. constructor concept is there for abstract classes	7. But we dont have any constructor concept for interfaces
8.Super most class is Object by default.	8.But we dont have any super most interface concept but we can extend and create Parent interfaces

9.absrtact class canbe inheirted using extends keyword	9. But interfaces canbe inheirted using implements keyword
10.we can write main() method	10. we cannot write main() method

Marked interface or tagged interface

-we can also declare an interface without any abstract methods which is called Marked interface or tagged interface
-the main advantage of Marked interfaces is giving an instruction to JVM to perform a special task.

Eg:

Cloneable, Serializable, EventListener,...

Packages

-Packages is a collection of class and interfaces in the form of .class files
-The main advantage of Packages is to improve maintenance of the application.
- we have following 2 types of packages in java
1. predefined packages
2. user defined packages

1. predefined packages

-Predefined packages are the packages which are given by sun micro systems or some other companies as a part of java
-we have following 3 types of predefined packages.

1. core packages

core packages are predefined packages given by sun micro systems which are begin with " java ".

2. extended packages

extended packages are also predefined packages given by sun micro systems which are begin with " javax ".

3. third party packages

third party packages are also predefined packages but which are given by some other companies as a part of java software.

Eg:

oracle.jdbc, com.mysql,...

Few example of core or extended packages:

java.lang:

this package is the collection of classes and interfaces using which we can perform basic operations like parsing, to store strings,.... This package by default available for every java program

Eg:

Object, String, Integer,...

java.io

this package is the collection of classes and interfaces using which we can perform basic input and output operations

Eg:

BufferedReader, PrintStream, FileReader,.....

java.util

this package is the collection of classes and interfaces using which we can display date, store the group of objects(Collection API),....

Eg:

Date, Collection,ArrayList,...

java.text

this package is the collection of classes and interfaces using which we can perform operations like formatting the date, number ,....

Eg:

NumberFormat, DateFormat,...

java.net

this package is the collection of classes and interfaces using which we can develop network related applications

Eg:

Socket, ServerSocket,...

java.sql

this package is the collection of classes and interfaces using which we can perform JDBC related operations like connecting to DB, creating the tables ,inserting records,...

Eg:

Connection, Driver, DriverManager,....

java.awt

this package is the collection of classes and interfaces using which we can develop GUI based Applciations

Eg:

Frame, Label,Button,...

javax.swing

this package is the collection of classes and interfaces using which we can develop a better GUI based Applciations

Eg:

JFrame, JLabel,JButton,...

java.applet:

this package is the collection of classes and interfaces using which we can develop an Applciations that runs in the browser

Eg:

Applet, AppletContext,...

2. user defined packages

-in java we can also create user defined packages according to our requirement.

-to create the user defined package we have to use a java keyword called " package "

-user defined packages contains only user defined classes or interfaces in the form of .class files.

syntax:

```
package   packagename;
          or
package   package1.package2.package3;
```

Rules:

1.While writing the package name we can specify packages in any number of levels but specifying one level is mandatory.

2.package statement must be written as first executable statement in the program.

3.we can write at most one package statement in the program

// program to Demo on packages

Wish.java

```
package com.inetsolv.corejava;
class Wish{
  public static void main(String args[]){
     System.out.println("Have A Nice Day @ InetSolv");
  }
}
```

compiling the program

```
javac -d . Wish.java
        or
javac -d E: Wish.java
        or
javac -d D:\practice Wish.java
```

- d this option is used to create the package based on name specified in the program using package statement and locate the generated class into this package.

. or E: or D: specify the location where to locate the created package

Execute the program

```
> java com.inetsolv.corejava.Wish
o/p:
  Have A Nice Day @ InetSolv
```

Note:

In java, for every java program following 2 packages are available by default.

1. java.lang
2. current working directory

we can use all the classes and interfaces are available in the above 2 packages but if we want to use classes and interfaces that are available in the other packages we must import into our program by using any one of the following 2 ways

1. using Fully Qualified Name 2. using import statement

1. using fully Qualified Name

- Fully Qualified Name means writing the class name using packagename directly. using this concept we can import only one class at a time.
-This concept will increase the code size and decrease the readability of the program and this concept is not recommended to use.

syntax:

```
class ReadingData{
public static void main(String args[]) throws java.io.IOException{
java.io.BufferedReader br = new java.io.BufferedReader(new java.io.InputStreamReader(System.in));
}
}
```

2. using import statement

using import statement we can import either only one class or many classes from the package.

we have following 2 types of import statements

1.implicit import

by using implicit import we can import many classes from the package.

syntax

```
import package.*;
import package1.package2.*;
```

Eg:

```
import java.io.*;
class ReadingData{
public static void main(String args[] throws IOException{
BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
}
}
```

here compiler will decide what class to be loaded in program hence it is called as implicit import.

2. explicit import (recommended)

by using explicit import we can import only one class from the package.

syntax:

```
import package.ClassName;
import package1.package2.ClassName;
```

Eg:

```
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.IOException;
class ReadingData{
public static void main(String args[]) throws IOException{
BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
}
}
```

here programmer is specifying explicitly what classes to be loaded hence it is called as explicit import

Note:

it is always recommended to use explicit import in the industry because it always improve the readability of the program

Rules for writing import statement

1. import statement must be written after the package statement and before the class declaration.
2. we can write any number of import statements

difference between implicit import and explicit import

using implicit import or explicit import both are same but explicit import will take a little bit more compilation time compared to implicit import but at run time there is no difference.

what is the difference between #include and import statements

#include statement is used in C, C++ languages to import the library functions into the program. this # inlcude statement will import the library functions into the program directly so that length of the program will be increased and compilation time is increased as well. #include will load all libray functions whether we use or not. this kind of loading is called static loading.

import statement used in java to import the classes from the packages. it will load the classes into a method area not into the program so that no code size will be increased and no compilation time is increased. this import statement will load only the classes what we use in the application this kind of loading is called as dynamic loading or load on demand.

Note:

when we import the main package then we can use only the classes available in the particular main package, but if we want to use the classes available in the sub package again we have to write one more import statement for sub package.

Eg1:

```
import java.*;                    // X- invalid
  class Demo{
    ArrayList al = new ArrayList();
  }
```

Eg2:

```
import java.util.*;                    // valid
  class Demo{
    ArrayList al = new ArrayList();
  }
```

Math class

- This class is mainly used to perform the different types mathmatical calculations like pow(), sqrt(), sin()...
- all methods of Math class are declared as static so that we can access all these methods directly using class name.

```
//wap to demo on Math class
class MathDemo{
public static void main(String args[]){
 System.out.println(Math.min(10,2));
 System.out.println(Math.max(10,2));
 System.out.println(Math.sqrt(2));
```

```
 System.out.println(Math.pow(5,2));
 System.out.println(Math.cos(0));
 System.out.println(Math.log(102));
 System.out.println(Math.floor(10.234));
 System.out.println(Math.ceil(10.234));
 System.out.println(Math.PI);
}
}
```

static imports

static imports are special kind of import statements given in java 1.5 version which are used to import the static members of any class into the program so that we can access those static members directly without any class name or object.

syntax:

importing all static members from particular class

```
import static package.ClassName.*;
import static  package1.package2.ClassName.*;
```

importing particular static member from particular class

```
import static package.ClassName.membername;
import static  package1.package2.ClassName.membername;
```

```
//wap to demo on static imports
import  static  java.lang.Math.max;
import  static  java.lang.Math.PI;
import  static  java.lang.Math.*;
import  static  java.lang.Integer.*;
class StaticImport1{
public  static  void main(String  args[]){
System.out.println(max(10,2));
System.out.println(min(10,2));
System.out.println(sqrt(2));
System.out.println(pow(10,2));
System.out.println(cos(0));
System.out.println(floor(10.945));//10
System.out.println(random()*100);
System.out.println(PI);
System.out.println(MIN_VALUE);
}
}
```

difference between normal import and static import

normal import is used to import the classes from packages but static imports are used to import the static members of the class.

Note:

but this static import concept is not recommended in real time applications , because it will reduce readability, some times compiler will get confuse (ambiguity) to use one property or method which is available in both classes.

```
import  static  java.lang.Byte.*;
import  static  java.lang.Integer.*;
class StaticImport2{
public  static  void main(String  args[]){
System.out.println(MAX_VALUE);
}}
```

Access specifiers

- Access specifiers are access modifiers or java keywords which are used to specify the scope or accessibility of members of the class and accessibility of class from a package.
- using Access specifiers we can provide security for our data
- we have following 4 Access specifiers in java with 3 keywords
  1. public
  2. private
  3. protected
  4. default (not a keyword)

1. public

-if any property is declared as public that can be accessible in all locations.

-we can apply this access specifier for any class , interfaces, methods, variables.

2. private

-if any property is declared as private that can be accessible only with in the same class where it is declared.

- we can apply this access specifier for any inner classes,inner interfaces,methods, variables

3. protected

-if any property is declared as protected then it can be accessible in same class,in sub class or non-sub class of same package , in sub class of other package but we can not access protected properties with in non-sub class of other package

- we can apply this access specifier for any inner classes,inner interfaces, methods, variables

4. default (not a keyword)

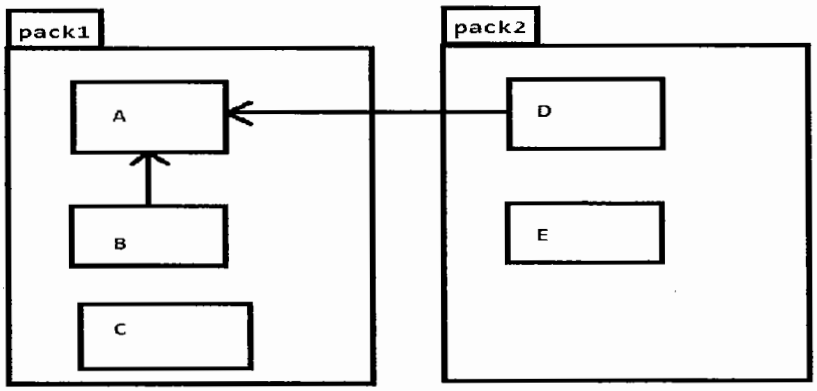
- if any property is declared without writing any public or protected or private keyword then these properties will have default access specifier.

-these properties can be accessible with in the same class,in sub class or non-sub class of same package, but we can not access these properties with in sub class of other package or non-sub class of other package.

-this default access specifier can also be called as package level access specifier. we can apply this access specifier for any classes, interfaces, methods, variables

```
package  pack1;

public class A{
   int x
}

class B extends A{

}

class C{
}
```

```
package  pack2;
import  pack1.*;
class D extends A{
}

class E {
}
```

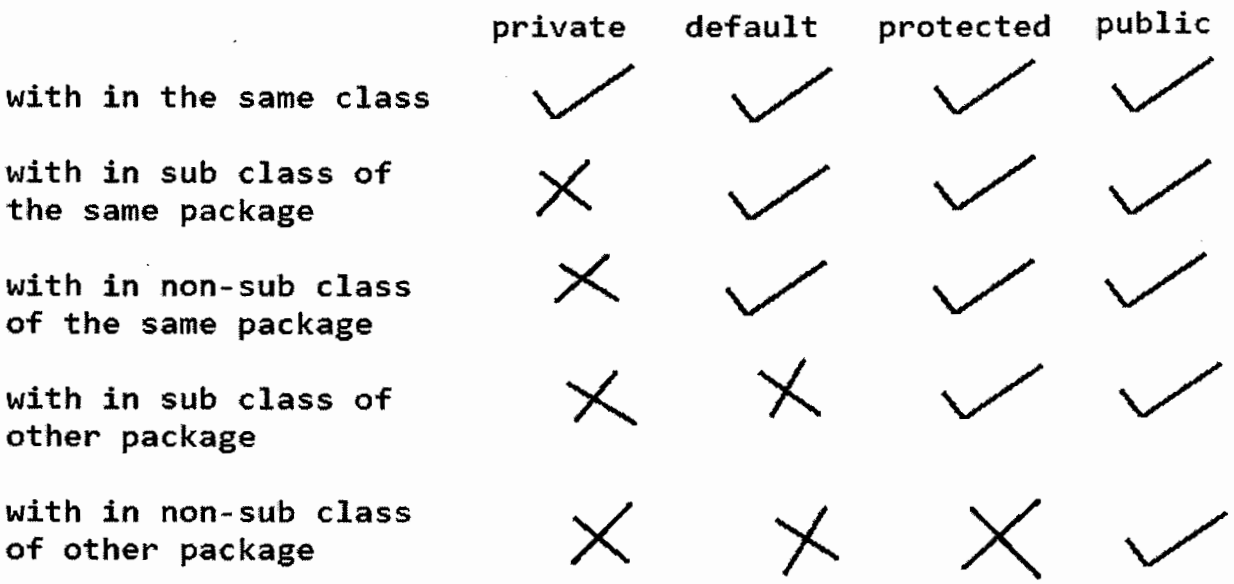


-if variable x is public then it can be accesible in A,B,C,D,E

-if variable x is private then it can be accesible only in A

-if variable x is protected then it can be accesible in A,B,C,D

-if variable x is default then it can be accesible in A,B,C



more accessible access specifier is public and more restricted access specifier is private.

Overriding Rule

- In Method overriding the 2 methods are declared with same method signature in 2 different classes which contains Is-A relation
- in Method overriding method signature must be same means 2 methods must contain same method name, same return type, same number of parameters or same type of parameters or same order of parameters
- in Method overriding access specifiers will also play the role.
- If we are method overriding the parent class method in child class then parent class method scope or access specifier can be promoted inside the child class but it can not be demoted.

access specifier promoting order

default<protected<public

Note:

in every interface all the methods are bydefaullt public abstract so that when ever we implement the interface then all its methods must be override using public access specifier only.

Illigal combination of access modifiers for a method

we never declare a method as abstract and private because abstract keyword says the method must be override with in the child class but private methods can not be inherited and cant be override.

abstract private void area(); X – invalid

Note:

-In Java program we can declare any number of classes which can be either public class or default class
- we can create any number of default classes
-but we can create at most 1 public class in a single program.
-Program name can be decided based on class declaration but program name can't be decided based on main() method
- If class is declared as default class then the program name can be anything.
- but if class is declared as public then program name must be same as class name.
Eg:

Demo.java

```
class A{
public static void main(String args[]){
System.out.println(" class A main() method");
}
}
class B{
public static void main(String args[]){
System.out.println(" class B main() method");
}
}
class C{
public static void main(String args[]){
 System.out.println(" class C main() method");
 }
}
class D{
}
```

compilation:

javac Demo.java

-when we compile any java program java compiler will generate .class file for every class avaiable in the program
-here if we compile above program then compiler will generate following 4 .class files

A.class
B.class
C.class
D.class

Execution

```
 > java A
  o/p:
  class A main() method
>java B
  o/p:
  class B main() method
> java C
  o/p:
  class C main() method
> java D
  Exception: NoSuchMethodError : main
> java Demo
  Exception: NoClassDefFoundError
```

create a package to perform all arithmetical operations

Addtion.java

```
package com.inetsolv.maths;
public class Addition{
private int a;
private int b;
public Addition(int a,int b){
this.a=a;
this.b=b;
}
public void add(){
System.out.println("Addition="+(a+b));
}}
```

Subtraction.java

```
package com.inetsolv.maths;
public class Subtraction{
private int a;
private int b;
public Subtraction(int a,int b){
this.a=a;
this.b=b;
}
public void sub(){
System.out.println("Subtraction="+(a-b));
}}
```

Multiplication.java

```
package com.inetsolv.maths;
public class Multiplication{
private int a;
private int b;
public Multiplication(int a,int b){
this.a=a;
this.b=b;
}
public void mul(){
System.out.println("Multiplication="+(a*b));
}}
```

Division.java

```
package com.inetsolv.maths;
public class Division{
private int a;
private int b;
public Division(int a,int b){
this.a=a;
this.b=b;
}
public void div(){
System.out.println("Division="+(a/b));
}}
```

compiling, generating .class file and locating into particular package

```
javac -d . Addition.java
javac -d . Subtraction.java
javac -d . Multiplication.java
javac -d . Division.java
// Write a Program to use the package
```

UsingPackage.java

```
import com.inetsolv.maths.Addition;
import com.inetsolv.maths.Subtraction;
import com.inetsolv.maths.Multiplication;
import com.inetsolv.maths.Division;
class UsingPackage{
public static void main(String args[]){
Addition a = new Addition(10,20);
Subtraction s = new Subtraction(10,2);
Multiplication m = new Multiplication(10,3);
Division d = new Division(10,3);
a.add();
s.sub();
m.mul();
d.div();
}
}
```

assignment

```
// write the above program using implicit import
```

Single tone class (single tone design pattern)

- single tone class is a class which allow us to create only one object at any time
- If we want to create a single tone type of class we have to follow following 2 rules
  1. It must contain a private constructor
  2. It must contain a factory method

factory method(factory method design pattern)

- A factory method is a method which is responsible for creating and returning the object of Its class.
- If we want to create a factory method we have to follow following 2 rules
  1. its return type must be its class type
  2. it must be a static method

```
//program to design a singleton class
class Demo{
static Demo d=null;
/*private constructor*/
private Demo(){
}
/*factory method*/
static Demo getObject(){
if(d==null){
 d = new Demo();
}
return d;
}
}
class SDPatternDemo1{
```

```
public  static void main(String  args[]){
 Demo  d1 = Demo.getObject();
 Demo  d2 = Demo.getObject();
 System.out.println(d1==d2);//true
//Demo d = new Demo(); -invalid
}
}
```

(or)

```
class  Demo{
private static  Demo d=new Demo();
private Demo(){
}
static  Demo  getObject(){
return  d;
}
}
class SDPatternDemo2{
public  static void main(String  args[]){
 Demo  d1 = Demo.getObject();
 Demo  d2 = Demo.getObject();
 System.out.println(d1==d2);//true
 }
}
```

Note:

1. we can not create an object for a class which contains a private constructor

```
    class  Demo{
       private  Demo(){
        }
     }
  class  Test{
   public  static void main(String  args[]){
      Demo  d= new Demo();                X - invalid
   }
  }
```

2. we can not inherit from a class which contains a private constructor because every child class constructor invokes it parent class constructor by default, but here parent class constructor is private.

```
   class   Demo{
      private Demo(){
        }
     }
   class   Test extends Demo{    X-invalid
      Demo(){
        }
     }
```

wrapper classes

- wrapper classes are used to convert the primitive values into object, this procedure is called as boxing
- For every primitive data type there is a corresponding wrapper class
- all wrapper classes available in java.lang package

Data type	Wrapper class
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double
char	Character
boolean	Boolean

constructors of Wrapper classes

- to create an object for any Wrapper class we have to use constructors of Wrapper classes.
- In all most all wrapper classes we have following 2 constructors
  --first one will take primitive value directly
  --and second one will take primitive value in the form of String

Eg:

```
Byte  b = new Byte(byte);
Byte  b = new Byte(String);

Short  s = new Short(short);
Short  s = new Short(String);

Integer  i = new Integer(int);
Integer  i = new Integer(String);

Long l = new Long(long);
Long l = new Long(String);

Float  f = new Float(float);
Float  f = new Float(double);
Float  f = new Float(String);

Double d = new Double(double);
Double d = new Double(String);

Character  c  = new Character(char);
Character  c  = new Character(String); X - not available
Boolean  b = new Boolean(boolean);
Boolean  b = new Boolean(String);
```

```
//wap to demo on Wrapper  classes
class WrapperClass1{
  public  static void main(String  args[]){
    int a=10;
    Integer p = new Integer(a); //boxing
```

```
    System.out.println(p);
    String b="120";
    Integer q = new Integer(b); //boxing
    System.out.println(q);
  }
}
```

-while creating an object for Boolean Wrapper class if constructor takes primitive boolean value as true it stores true otherwise if it takes primitive boolean value as false it stores false.

-while creating an object for Boolean Wrapper class if constructor takes primitive boolean value in the form of string then if string value is true in any case (true, True,TRUE,...) then it stores true otherwise if string value is other than true it stores false.

Eg:

```
Boolean b1 = new Boolean(); //CTE
Boolean b2 = new Boolean(true); //true
Boolean b3 = new Boolean(false); //false
Boolean b4 = new Boolean(True); //CTE
Boolean b5 = new Boolean("TRUE"); //true
Boolean b6 = new Boolean("false"); //false
Boolean b7 = new Boolean("suresh"); //false
Boolean b8 = new Boolean(null); //false
Boolean b9 = new Boolean(1); //CTE
```

Note:

in java we have 3 predefined or implicit literals (values) i.e, true,false,null which are given in lower case and they are not java keywords.

Methods of Wrapper class

1. valueOf():

This methd is also used to convert the primitive value into object(boxing)

syntax1:

public static WrapperClass valueOf(primitive)

--this method is available in all the 8 wrapper classes.

syntax2:

public static WrapperClass valueOf(String)

-- this method is available in all the wrapper classes except Character class.

Eg:

```
class WrapperClass2{
public static void main(String args[]){
int a=10;
Integer p = new Integer(a); //boxing using constructor
Integer q = Integer.valueOf(a);//boxing using valueOf()
System.out.println(p);
System.out.println(q);
String b="120";
Integer x = new Integer(b); //boxing using constructor
Integer y = Integer.valueOf(b); //boxing using valueOf()
System.out.println(x);
System.out.println(y);
}
}
```

2. primitivetype xxxValue()

this method return the primitive value available in the respective object, it means we are converting object into primitive value this procedure is called as unboxing.

byteValue()
shortValue()
intValue()
longValue()
floatValue()
doubleValue()
} Byte, Short, Integer, Long, Float, Double

charValue() } Character

booleanValue() } Boolean

3. static primitiveType parseXXX(String)

- this method converts the primitive value available in the form of String into required primitive type.
- this procedure is called as parsing.(kind of unboxing)
- this method is available in all wrapper classes except Character class.

Eg:

```
class WrapperClass3{
public static void main(String args[]){
int a=1000;
Integer i1 =Integer.valueOf(a);//boxing
int  v1 = i1.intValue();//unboxing
byte  v2 = i1.byteValue(); //unboxing
System.out.println(v1);
System.out.println(v2);
//parsing
String b="20";
int  v3 = Integer.parseInt(b);
System.out.println(v3);
}
}
```

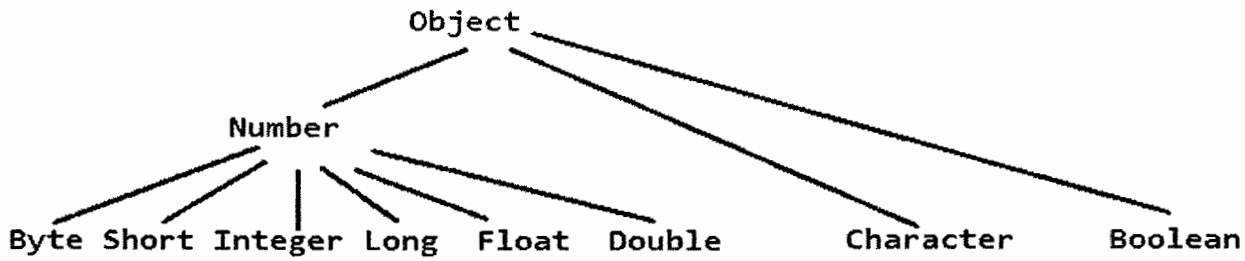
4. String toBinaryString(int)/toHexString(int)/toOctalString(int)

these methods are used to convert given the number into either binary number system or hex-decimal number system or octal number system

Eg:

```
class WrapperClass4{
public static void main(String args[]){
int a=10;
System.out.println(Integer.toBinaryString(a)); //1010
System.out.println(Integer.toHexString(a)); //a
System.out.println(Integer.toOctalString(a)); //12
}
}
```

Hierarchy of Wrapper classes



typecasting

typecasting is a process of converting one primitive type to another primitive type or one reference to another reference type

parsing

parsing is a process of converting String into corresponding primitive type

boxing

converting a primitive value into object is called boxing. from java 1.5 onwards this procedure is done automatically by compiler hence it is called auto boxing.

unboxing

converting an object value into primitive type is called un boxing. from java 1.5 onwards this procedure is automatically done by compiler hence it is called auto un boxing

Eg:

```
Integer a,b;
a= new Integer(10); //mannual boxing
        or
a=10;  //autoboxing
b=20; //autoboxing
int c = a.intValue()+b.intValue(); //mannual unboxing
        or
int c =a+b; //autounboxing
```

Object class

- Object class is a predefined class available in java.lang package
- Object class is called supermost class in java language because in java every user defined class or predefined classes are either directly or indirectly extending from Object class.
- Because of this reason the methods of Object class we can access in any class directly without creating any object.
- Object class contains some usefull methods which are required by every class in java.

Methods of Object class

1. String toString()

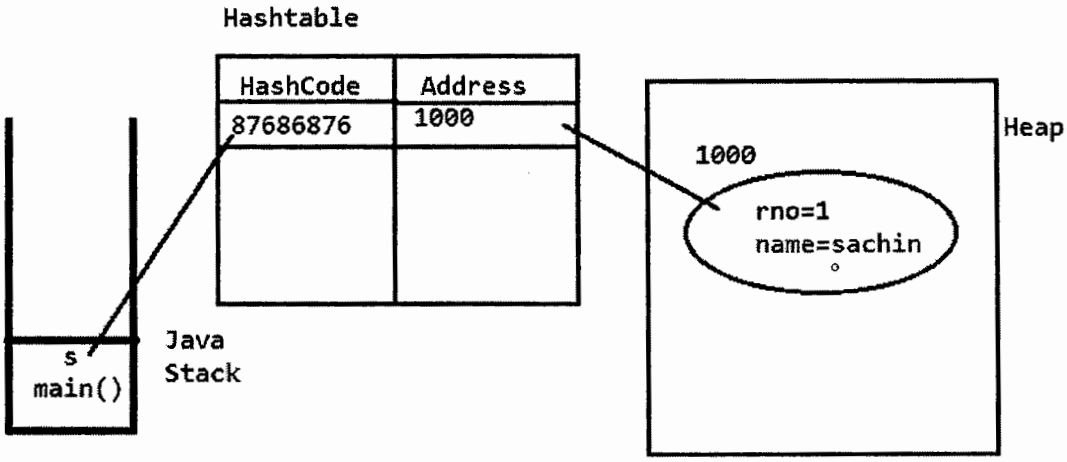
this method used to represent the object in the form of string. when we display any object it will call toString() method automatically whether we specify or not.

```
//program to demo on toString()
class Student{
 int rno;
 String name;
 Student(int rno,String name){
  this.rno=rno;
  this.name=name;
 }}
```

```
class ObjectDemo1{
 public  static void main(String  args[]){
  Student  s = new Student(1,"sachin");
  System.out.println(s); //Student@19821f
  System.out.println(s.toString()); //Student@19821f
  Student s1 = new  Student(1,"sachin");
  System.out.println(s1);//Student@addbf1
  }
}
```

Note:

when ever an object is created to hold the address of corresponding object there will be given a unique identification number called hash code.



predefined code of toString() method in Object class

```
public String toString() {
  return getClass().getName() + "@" + Integer.toHexString(hashCode());
   }
```

when we display the object of the class it will display hashcode of the corresponding object but if we display hashcode user get confuse and can not understand that hashcode, if we want to display object data instead of object hashcode we have to override the toString() method inside the particular class.

Eg:

```
class  Student{
 int rno;
 String name;
Student(int rno,String name){
this.rno=rno;
this.name=name;
}
public  String  toString(){
return  rno+"-"+name;
}
}
class ObjectDemo11{
public  static void main(String  args[]){
  Student  s1 = new Student(1,"sachin");
   System.out.println(s1); //1-sachin
```

```
  Student s2 = new  Student(2,"sehwagh");
   System.out.println(s2);//2-sehwagh
  String  str = new String("java");
  System.out.println(str);
  Integer i = new Integer(10);
  System.out.println(i);
}
}
```

Note:

- String, StringBuffer,Wrapper classes are already overriding toString() method.

2. Class getClass()

- this method returns the object in the form of Class using which we can get the information of particular class like class name, belongs what package,.....
- when ever we want to know what it is underlying class name of the particular object we have to use getClass() method of Object class.

Eg:

```
class  Student{
 int rno;
 String name;
Student(int rno,String name){
 this.rno=rno;
 this.name=name;
 }
}
class ObjectDemo2{
public  static void main(String  args[]){
  Student  s = new Student(1,"sachin");
  Class  c =  s.getClass();
System.out.println(c);
System.out.println(c.getName());
System.out.println(c.getSuperclass());
}
}
```

note:

lower case word class is a java keyword where capital Class is a predefined java class which is used to get the information of anyuse defined class or any predefined class.

3. int hashCode()

- this method returns hashcode of the particular object .
- an hashcode is a unique identification number which holds address of the corresponding object.

```
//program to demo on hashCode()
class  Student{
 int rno;
 String name;
Student(int rno,String name){
this.rno=rno;
this.name=name;
}
}
class ObjectDemo3{
```

```
public  static void main(String  args[]){
 Student  s1 = new Student(1,"sachin");
 Student  s2 = new Student(2,"dhoni");
System.out.println(s1.hashCode());//1671711
System.out.println(s2.hashCode());//11394033
}
}
```

Note:

-we can also provide our own hashcode then we have to override the hashcode() method inside corresponding class

Eg:

```
class  Student{
static  int count=1;
 int rno;
 String name;
Student(int rno,String name){
this.rno=rno;
this.name=name;
}
public  int hashCode(){
 return  count++;
}
}
class ObjectDemo33{
public  static void main(String  args[]){
 Student  s1 = new Student(1,"sachin");
 Student  s2 = new Student(2,"dhoni");
 System.out.println(s1.hashCode()); //1
 System.out.println(s2.hashCode()); //2
 }
}
```

4. boolean equals(Object o)

this method compares the 2 references whether they contain same object or not bydefault. but if we want to compare the equality of the content of 2 objects then we have to override equals() method with in the particular class.

```
//program without overriding equals()
class  Fruits{
String name;
double  price;
Fruits(String name,double price){
this.name=name;
this.price=price;
}
}
class ObjectDemo4{
public  static void main(String  args[]){
 String  str1 = new String("java");
 String  str2 = new String("java");
System.out.println(str1.equals(str2)); // true
 Fruits  f1 = new Fruits("Apple",35.0);
 Fruits  f2 = new Fruits("Apple",35.0);
```

```
 System.out.println(f1.equals(f2)); // false (reference checking)
}
}
```

Note:

-In String class already equals() method is overridden in such a way that it will compare the equality of the content of 2 Strings.

```
//program with overriding equals() method
class Fruits{
String name;
double price;
Fruits(String name,double price){
this.name=name;
this.price=price;
}
public boolean equals(Object o){
  Fruits f = (Fruits) o; // typecasting ( downcasting)
if(this.name.equals(f.name) && this.price==f.price){
 return true;
}
else{
 return false;
}
}}
class ObjectDemo44{
public static void main(String args[]){
 Fruits f1 = new Fruits("Apple",35.0);
 Fruits f2 = new Fruits("Apple",35.0);
 System.out.println(f1.equals(f2)); // true (content checking)
}
}
IQ:
class Test{
public static void main(String args[]){
 StringBuffer sb1 = new StringBuffer("java");
 StringBuffer sb2 = new StringBuffer("java");
  System.out.println(sb1.equals(sb2)); //false
  }
}
```

Note:

Here StringBuffer class is not overriding equals() method so that in the above program when we compare 2 StringBuffer objects using equals() method it will check reference of 2 objects but not equality of the content.

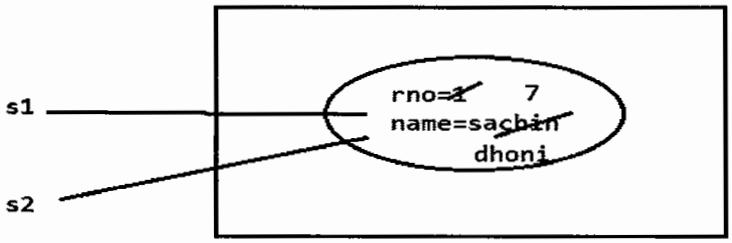
5. Object clone():

this method used to clone or copy the object so that we can take the backup for the object, but in java every object is created in a way that, it can not be copied directly, if we want to perform this special operations we must follow following rules.

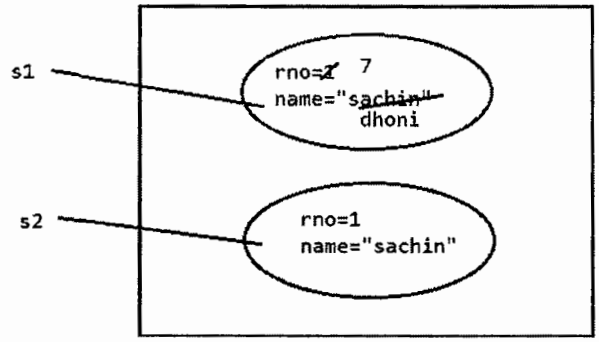
1. the cloned object must be type casted to corresponding object
2. we must handle an exception called CloneNotSupportedException
3. corresponding class must implement a marked interface called Cloneable interface

```
//program without cloning (references are copied)
```

```
class  Student{
int rno;
String name;
Student(int rno,String name){
this.rno=rno;
this.name=name;
}
public  static void main(String args[]){
 Student  s1 = new Student(1,"sachin");
 System.out.println(s1.rno+"\t"+s1.name);  // 1   sachin
 Student  s2 = s1; //shalow  cloning or partial cloning
 System.out.println(s2.rno+"\t"+s2.name);  // 1   sachin
 s2.rno=7;
 s2.name="dhoni";
 System.out.println(s1.rno+"\t"+s1.name); //7   dhoni
 System.out.println(s2.rno+"\t"+s2.name); //7   dhoni
 }
}
```




```
// program with cloning
class  Student implements Cloneable{
int rno ;
String name;
Student(int rno,String name){
this.rno=rno;
this.name=name;
}
public  static void main(String args[]) throws CloneNotSupportedException{
 Student  s1 = new Student(1,"sachin");
 System.out.println(s1.rno+"\t"+s1.name); //1   sachin
 Student  s2 = (Student)  s1.clone(); //deep  cloning
 System.out.println(s2.rno+"\t"+s2.name); //1   sachin
 s1.rno=7;
 s1.name="dhoni";
 System.out.println(s1.rno+"\t"+s1.name); //7   dhoni
 System.out.println(s2.rno+"\t"+s2.name); //1   sachin
 }
}
```



assignment

//write the above progam by writing main() method in a different class

reflections

- reflections is the concept of self checking the class or getting the information of class like what is the name of the class, what is the super class name,all its methods,constructors,...

-all reflection related classes are available in java.lang.reflect package.

```
//wap to demo on reflections
import java.lang.reflect.*;
class Vehicle{
}
class Car extends Vehicle{
public int capacity;
public String model;
public double price;
public Car(){
}
public Car(int capacity){
}
public Car(String model){
}
public Car(double price){
}
public void action1(){
}
public void action2(){
}
public void action3(){
}
}
class ReflectDemo{
public static void main(String args[]) throws ClassNotFoundException{
/*
//proc-1
Car c = new Car();
Class cl = c.getClass();
*/

/*
```

```
//proc-2
Class cl = Class.forName("java.lang.String");
*/

//proc-3
Class  cl = Car.class; //.class is extenson name
System.out.println(cl.getName());
System.out.println(cl.getSuperclass());
System.out.println(cl.getPackage());
Field   flds[] = cl.getFields();
for(Field f:flds){
 System.out.println(f);
}
Constructor  cons[] = cl.getConstructors();
for(Constructor co:cons){
 System.out.println(co);
}
Method  methods[] =   cl.getMethods();
for(Method m:methods){
 System.out.println(m);
}
}
}
```

Note:

Class.forName() method used to load the classes into JVM explicitly. if the specified class is not available Class.forName() method throws a runtime Exception called ClassNotFoundException

```
//creating an object of the class using Class.forName()
class  Student {
int rno;
String name;
void display(){
 System.out.println(rno+"\t"+name);
}
}
class  Testng{
public static void main(String args[]) throws ClassNotFoundException,InstantiationException,IllegalAccessException{
Class cl = Class.forName("Student");
Student s = (Student) cl.newInstance();
  s.rno=1;
  s.name="sachin";
  s.display();
}}
```

enum keyword

enum is a java keyword introduced in java 1.5 which is used to create the named constants.

syntax:

```
 enum   EnumName{
   //group of constants
  }
```

Eg:

```
 enum  Day{
   SUN,MON,TUE,WED,THU,FRI,SAT
  }
```
- java compiler will create .class for every class or interface or enum available in the program.

compilation

javac Day.java

Displaying profile

```
   javap Day
final class Day extends java.lang.Enum{
  public static final Day SUN;     //Day SUN = new Day();
  public static final Day MON;
  public static final Day TUE;
  public static final Day WED;
  public static final Day THU;
  public static final Day FRI;
  public static final Day SAT;
  public static Day[] values();
  public static Day valueOf(java.lang.String);
  static {};
}
```

Note:

-internally enum is created as final class that extends predefined abstract class called Enum.

-all constants are public static final

-every constant hold an object of corresponding enum.

-java enum is very efficient compared to c-language enum because in c-language enum contains only constatnts but in java enum can contain constants along with other code like constructors, methods, ...

Rules

1. an enum can be empty
2. an enum can contain only constants
3. an enum can contain constants along with othercode
4. an enum can not contain only othercode without constants
5. terminating constans by ; is optional when there is no other code is not avaialable otherwise if othercode is available then terminating constans by ; is mandatory
6. when we write othercode writing constants must be the first line
7. we can also write the main() method in enum

Eg:

```
enum  Day{
  SUN,MON,TUE,WED,THU,FRI,SAT;
   public  static  void main(String args[]){
    System.out.println("this is enum");
   }
 }
```

8. enum types can not be instantiated that is object can not be created for enum

Eg:

```
  Day  d=  new Day();   X invalid
```

9.we can write contructor inside the enum which can be accessed by creating a constant in enum

Eg:

```
enum  Day{
   SUN,MON,TUE,WED,THU,FRI,SAT;
Day(){
  System.out.println("this is constructor");
 }
 public  static  void main(String args[]){
  System.out.println("this is main()");
   }
 }
```

10. we can write instance variables, instance methods,... in enum

Eg:

```
enum  Fruits{
APPLE,MANGO(100),BANANA,MELON(200),ORANGE,GRAPES(15);
int price;
Fruits(){
price=39;
}
Fruits(int price){
this.price=price;
}
int getPrice(){
return price;
}
}
class Enum1{
 public  static void main(String args[]){
Fruits f = Fruits.BANANA;
 System.out.println(f+"\t"+f.getPrice());
 Fruits  frs[] = Fruits.values();
for(Fruits fr:frs){
  System.out.println(fr+"\t"+fr.getPrice());
  }
 }
}
```

Note:

when we write constants in enum they will invoke only 0 parameterized constructor, but if we want to invoke any parameterized constructor we have to write the constant by passing some parameters.

11.

-up to java 1.4 version we can pass byte/short/char/int datatype values into switch-case

-but in 1.5 version onwaords we can pass primitive datatypes byte/short/char/int and its corresponding Wrapper classes like Byte/Short/Character/Integer

- from java 1.5 onwards we can also pass enum as argument in switch-case

-in java 1.7 we can also pass a String into switch-case as a argument

Eg:

```
enum  Day{
  SUN,MON,TUE,WED,THU,FRI,SAT;
   }
class Enum3{
```

```
 public  static  void main(String args[]){
 //      Day  d = Day.TUE;
     Day  d = Day.valueOf("TUE");
 switch(d){
   case SUN :System.out.println("It is a joly Day");  break;
   case MON :System.out.println("First Working Day");  break;
   case TUE :System.out.println("Second Working Day");  break;
   case WED :System.out.println("Third Working Day");  break;
   case THU :System.out.println("Fourth Working Day");  break;
   case FRI :System.out.println("Fifth Working Day");  break;
   case SAT :System.out.println("It is shopping Day");  break;
//case ONE: System.out.println("Shopping Day");break;-invalid
    }
   }
}
```

12.

-an enum can not extend a class or enum, because enum internally created as a final class which extends predefined Enum class

-we can not create a class that extends any enum because internally enum is created as a final class and we can not extend a final class.

-an enum can implement any number of interfaces because internally enum is created as a final class and a class can extend a class and also implement any number of interfaces.

Exception Handling

- When ever we develop any application there is a chance of occurring errors in the application. As programmer we have to develop an application which does not contain any error
- we have following 2 types of errors

1. compile time errors

-Errors that occur at the time cimpilation of the program are called as compile time errors.

-Compile errors occured because if we dont follow the java syntaxes properly , java programming rules properly, ..

-compile time errors are identified by java compiler.

2. run time errors

-Errors that occur at the time execution of of the program are called as runtime errors.

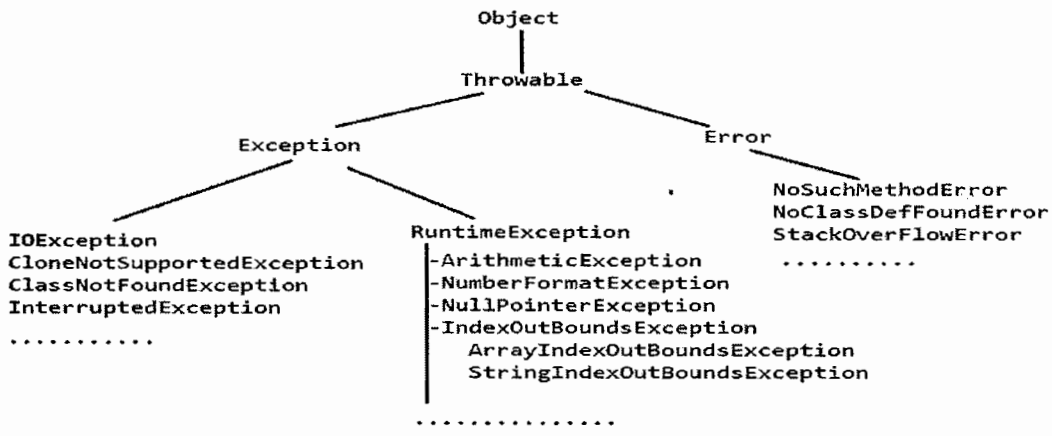
-Run time errors are also called as Exceptions

-Exceptions may occur because of programmer logic fails or jvm fails

-Exceptions are identified by jvm

-The process of handling these run time errors or exceptions is called as Exception Handling

Exception Hierarchy



Object	-Object is the super most class of all classes available in java
Throwable	-Throwable is the super most class of all Exceptions
Exception	-Exception is super most class of all exceptions which may occur because of programmer logic failure. These exceptions we can handle
Error	-Error is the super most class of all exceptions which occur because of jvm failure - These Errors we can not handle

Exceptions

-Exceptions are the run time errors which occur during the execution of the program.

-Exceptions may occur because of bad input values, connecting file is not available,...

-We have following 2 types of Exceptions

1. compile time exceptions

-Exceptions that identified at compilation time and occurred at runtime are called as compile time exceptions.

-These compile time Exceptions are also called as **checked exceptions**

-An exception said to be checked exception whose exception handling is mandatory as per the compiler.

Eg:

IOException, ClassNotFoundException, CloneNotSupportedException....

2. runtime exceptions

-Exceptions that identified and occured at run time are called as runtime exceptions.

-These runtime Exceptions are also called as **unchecked exceptions**

-An exception said to be unchecked exception whose exception handling is optional as per the compiler.

Eg:

ArithmeticException, NumberFormatException, NoSuchMethodError,..

Note:

all child classes of Error and RuntimeException classes are called as unchecked exception and remaining classes are called as checked exceptions.

Exception Handling

- while executing the program there is a chance of occurring errors in the application which are called exception.

-when ever an exception is occurred then the program will terminated in the middle of the execution of the program. which is called abnormal termination or incomplete execution.

-To resolve this problem we have to use exception handling.

-Exception handling is the process of handling the statements or skipping the statements which may generate exception and providing alternative solution, continue with the rest of the program and close/terminate program normally.

```
// wap without exception handling
import java.io.*;
class ExceptionDemo{
public static void main(String[] ar) throws IOException{
BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
 System.out.println("Enter Any 2 Numbers");
 int a=Integer.parseInt(br.readLine());
 int b=Integer.parseInt(br.readLine());
 System.out.println("main() method started");
 System.out.println("Hi");
 System.out.println("How Are you");
 System.out.println("Addition="+(a+b));
 System.out.println("Division="+(a/b));
 System.out.println("Multiplication="+(a*b));
```

```
System.out.println("Subtraction="+(a-b));
System.out.println("k i have done everything");
System.out.println("main() method end");
}
}
```

```
output 1
--------------
Enter Any 2 Numbers
10
2                                   //nomal termination
main() method started
Hi
How Are you
Addition=12
Division=5
Multiplication=20
Subtraction=8
k i have done everything
main() method end


output 2
-----------------
Enter Any 2 Numbers
10                                 //abnormal termination
0
main() method started
Hi
How Are you
Addition=10
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at Demo.main(Demo.java:14)
```

internal procedure

- when ever any exception is occurred with in the particular method then corresponding exception object is created by the particular method and given to JVM, then JVM will search for exception handling code with in the particular method if not available it will search in its caller method even there is not available finally JVM will call " Default Exception Handler " Program which will display the Exception object and terminate the method from java stack. (abnormal termination)
- In java Exception Handling can be done by using following 5 java keywords
    1. try
    2. catch
    3. finally
    4. throw
    5. throws

1. try

this block contains the group of statements which may generate the exception.

syntax:

```
try{
   //group of statements which may generate the exception
}
```

2. catch

-this block is used to catch the corresponding exception that has occurred.

-in this catch block we can either display messages or provide exception handling code.

syntax:

```
catch(AnyException ae){
 // group of exception handling statements
}
```

3. finally

-this block contains the code which should be executed either exception may occur or not.

-generally this block contains the code for releasing the resources like closing the db connection, closing the file,...

syntax:

```
finally{
 // group of statements for releasing the memory
}
```

```
//Wap to demo on Exception Handling
import java.io.*;
class Exception2{
public static void main(String[] ar) throws IOException{
BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
 System.out.println("Enter Any 2 Numbers");
 int a=Integer.parseInt(br.readLine());
 int b=Integer.parseInt(br.readLine());
 System.out.println("main() method started");
 System.out.println("Hi");
 System.out.println("How Are you");
 System.out.println("Addition="+(a+b));
try{
 System.out.println("Division="+(a/b));
}
catch(ArithmeticException ae){
// System.out.println("Division="+(a/1));
// System.out.println("Hello Boss what Happned division by zero is //not possible otherwise goto maths tution....");
// System.out.println(ae.toString());
// System.out.println(ae.getMessage());
ae.printStackTrace();
}
finally{
 System.out.println(".....Thank You...");
}
 System.out.println("Multiplication="+(a*b));
 System.out.println("Subtraction="+(a-b));
 System.out.println("k i have done everything");
 System.out.println("main() method end");
}
}
```

we can display Exception or runtime error message using any one of the following 4 ways

1. using System.out.println():

using System.out.println() we can display our own customized message.

Eg:

```
try{
}
catch(AnyException ae){
  System.out.pritnln("Customized Message");
}
```

2. using toString():

if use toString() method it will display the reason why Exception Occured and Exception classname.

Eg:

```
        try{
        }
        catch(AnyException ae){
         //System.out.pritnln(ae);
         System.out.pritnln(ae.toString());
        }
```

3. using getMessage()

if we use getMessage() method it will display only the reason why Exception Occured.

Eg:

```
        try{
         }
        catch(AnyException ae){
         System.out.pritnln(ae.getMessage());
         }
```

4. using printStackTrace():

if we use printStackTrace() method it will display the Exception information in detail like reason, Exception class name, program name where it is occurred, in what method, in what line.

Eg:

```
        try{
         }
        catch(AnyException ae){
          ar.printStackTrace();
         }
```

Rules for writing try-catch-finally

1.try must followed by either catch block or finally block or both
2.catch must followed by either catch block or finally block
3.catch must preceded by either catch block or try block
4.finally must preceded by either catch block or try block
5. we can write only one finally statement and one try block and many catch blocks in try-catch-finally chain.
6. nesting try-catch-finally is also possible
7. when we write multiple catch blocks if Exceptions are not having any Is-A relation then we can write catch block in any order otherwise we must write catch blocks in a order like first child class and followed by parent class.
8. we can not write 2 catch blocks which are going to catch same exceptions.

How JVM execute try-catch-finally

-First it will execute the statements written before try-catch blocks next JVM Will enter into try block and if no exception occurred then it will execute all the statements of try block and it will skip all the catch block and control will be given after try-catch blocks

- JVM Will enter into try block and if exception occurred then it will skip remaining statements of try block and control will be given any one of the matching catch blocks and executes the statements available in the catch block and next control will be given after try-catch blocks

Note:

1.when Exception occurred then control will be given any of the matching catch blocks and the remaining catch blocks will be skipped. and control will be given after try-catch blocks

2. If no catch block is matching then JVM will call or invoke " DefaultExceptionHandler" and which always cause for abnormal termination.

3. If Exception Occurred or not in both the cases finally block will be executed.

4. throws

this keyword is used to transfer the responsibility of Exception handling to its caller method.

Eg:

```
class Exception3{
        static void m1(){
          System.out.println("first line in m1()");
           m2();
          System.out.println("last line in m1()");
         }
        public static void main(String args[]){
         System.out.println("first line in main()");
          m1();
        System.out.println("last line in main()");
        }
        static void m2() {
         System.out.println("first line in m2()");
          try{
             m3();
           }
         catch(ArithmeticException ae){
            System.out.println("Division="+10/1);
            }
        System.out.println("last line in m2()");
    }
        Static void m3() throws ArithmeticException{
        System.out.println("first line in m3()");
        System.out.println("Division="+10/0);
        System.out.println("last line in m3()");
        }
   }
```

Note:

it is always recommended to use try-catch-finally blocks to handle the exceptions. But it is not recommended to transfer the exception handling job to its caller method by using throws keyword which always leads to abnormal termination.

5. throw

-by default all predefined exceptions are created and thrown implicitly and identified by JVM

-but if we want to throw the exceptions explicitly then we have to use throw keyword.

Eg:

```
import java.io.*;
class Exception4{
public static void main(String args[]) throws IOException{
 System.out.println("first line in main()");
 int a,b,c=0;
```

```
  a= Integer.parseInt(args[0]);
  b= Integer.parseInt(args[1]);
   try{
   if(b==0){
    ArithmeticException  ae1 = new ArithmeticException("Division by                    by 0 is not
possible");
     throw ae1;
   }
   c=a/b;
   System.out.println("Division="+c);
  }
 catch(ArithmeticException ae){
   System.out.println(ae.getMessage());
  }
 System.out.println("last line in main()");
}
}
```

User defined Exceptions

-user defined exceptions are exceptions that are developed by programmer according to his requirement.

-if none of the predefined Exceptions are suitable for our application requirement then we have to go for user defined exceptions.

- to create the user defined exceptions we have to follow the following procedure

procedure:

1.all exception classes are extending from Exception class. So that to create user defined exception we should create a class that extends any one of the following 2 classes

Exception(Checked Exceptions) RuntimeException(Unchecked Exceptions)

2.create a parameterized constructor which calls super class parameterized constructor to set the Exception Message.

3. JVM don't know when to generate the user defined exception and how to create and throw the object for User defined exception so that we should explicitly create the object for User defined exception and throw manually using throw keyword.

Eg:

```
import java.io.*;
class AgeException extends  Exception{
  AgeException(String msg){
       super(msg);
 }}
class  ExceptionDemo5{
public  static  void  main(String  args[]) throws  IOException{
 BufferedReader br =  new  BufferedReader( new InputStreamReader(System.in));
  System.out.println("Enter Your Age");
  int  age =  Integer.parseInt(br.readLine());
 try{
  if(age<18){
   AgeException  ae = new  AgeException("Sorry You hav to wait for"+(18-age)+"Years");
      throw  ae;
        }
     System.out.println("Congratulations you are eligible for Voting...");
   }
   catch(AgeException  ae){
```

```
        System.out.println(ae);
      }
  }
  }
```

ADV JSE(CORE PART II)

Collections

Array:

An array is a collection of elements which are stared in continuous memory locations.

Limitations of array (or) disadvantages of array:

1.The size of an array is fixed it means once array size is decided it can't be increased or decreased. With this some times memory may be wasted or sometimes memory may not be sufficient.

2.There are no pre-defnined functions to perform the operations like Insertion,Deletion, Searching, Sorting etc. so that as a programmer we have to write our own logic.

3. arrays stores only a group of similar elements Because of the problems in an array the java people have come up with collection framework

Collections

-Collections concept introduced in java 1.2 version to resolve the problems of arrays

-Collections can hold a group of objects

-Collection size not fixed it means as we are inserting and deleting the elements the size of the collection dynamically increased or decreased.

-Every collection internally follows data structures and contains predefined functions it means we no need to write new logic for performing the operations like Insertion,Deletion, Searching, Sorting etc.

Collection object:

-An object is said to be collection object if it holds or stores a group of other objects.

-Collection never stores any primitive values,but if we want to store primitive value we have to represent primitive value as object

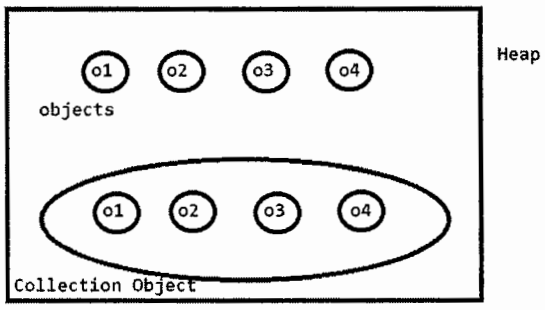
Collection classes

- Collection class is a class whose object is can store a group other objects

Eg:

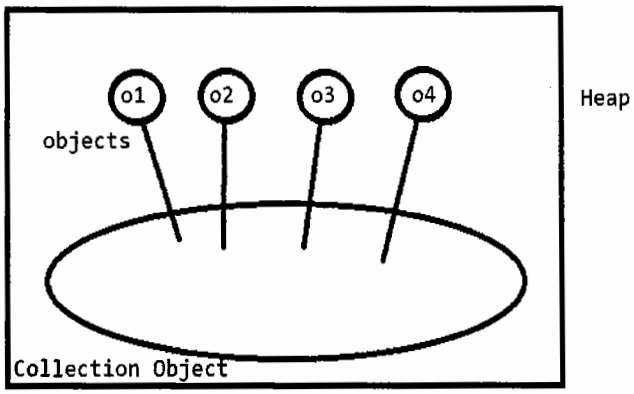
ArrayList, HashSet,HashMap,....

collection stores a group of objects like follows



- Here objects o1, o2, o3 and o4 are stored for 2 times there by we are wasting the memory with in the JVM.

-To save the memory with in the jvm when the objects are stored in the collection object, the jvm stores the references of the objects with in the collection objects instead of storing the copy of objects directly.



- All the collection classes are available in "java.util" (utility) package.
- All the collection interfaces and collection class and together as collection frame work.
- All the collection classes are classified into three categories

1. List:

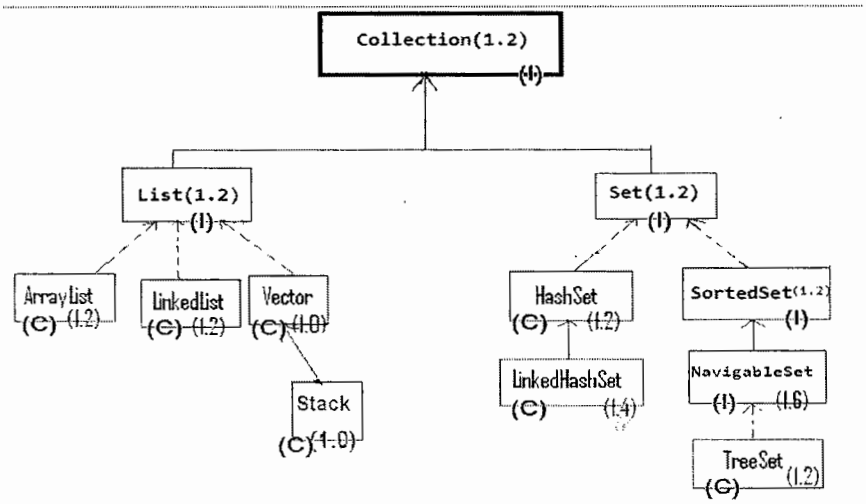
- This category is used to store group of individual elements where the elements can be duplicated.
- List is an Interface whose object can not be created directly.
-To work with this category we have to use following implementations class of list interface
ArrayList, Linked list, Vector, Stack

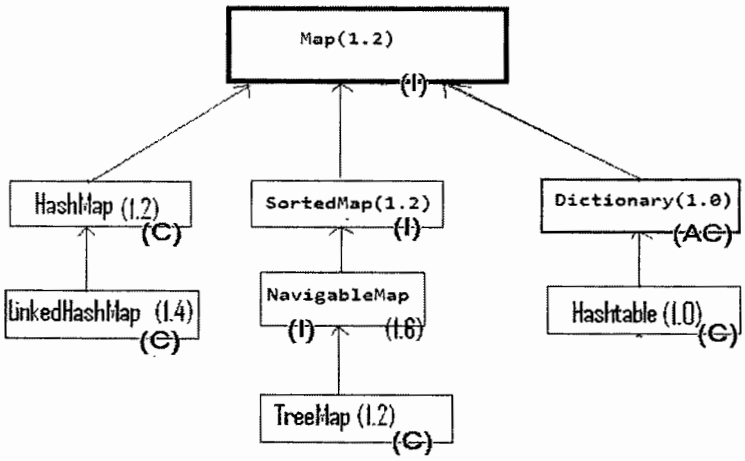
2.Set:

-This category is used to store a group of individual elements. But they elements can't be duplicated.
- Set is an interface whose object can not be created directly.
- To work with this category we have to use following implementations class of Set interface
HashSet, LinkedHashSet and TreeSet

3. Map:

- This category is used to store the element in the form key value pairs where the keys can't be duplicated, values can be duplicated.
- Map is an interface whose object can not be created directly.
-To work with this category we have to use following implementation classes of Map interface
HashMap, LinkedHashMap, TreeMap ,Hashtable





-Both List, Set are extending from Collection interface

- Map is part of Collection Framework but not extending from Collection interface

List category

ArrayList

-ArrayList is the implementation class of List interface which is used to store group of individual objects where duplicate values are allowed

-ArrayList internally follows array structure it means in ArrayList all the elements are stored in continuous memory locations same like an array but ArrayList size is not fixed

-ArrayList is not a synchronized class

-If any object is synchronized we can access only one thread at a time but if an object is not synchronized then we can access multiple threads

creation of ArrayList:

ArrayList<E> al= new ArrayList<E>();

this constructor is used to create the new ArrayList with default capacity 10 and the size of arraylist is 0.

ArrayList<E> al= new ArrayList<E>(int capacity);

this constructor is used to create the new ArrayList with specified capacity and the size of arraylist is 0.

Note:

-size means number of elements that are stored into Collection.

-capacity means the memory allocated for elements.

ArrayList<E> al= new ArrayList<E>(Collection obj);

this constructor is used to create the new ArrayList by copying the elements of any existing Collection (List,Set).

here

<E> - is called generic parameter type

E - Element type which must be reference type but not any primitive type

Methods of ArrayList

1. boolean add(E obj)

this method adds new element to the end of the List

2. void add(int position,E obj)

this method insert new element into the List at specified position

3.boolean remove(E obj)

this method removes specified element from the List(mannual boxing is needed)

4.E remove(int position)

this method removes particular element from the List at specified position.

5. int size()

this method returns the count of the number of element available in the List.

6. void clear()

this method removes all the elements from the List

7. boolean isEmpty()

this method returns true if List does not cotain any element otherwise it returns false

8. boolean contains(E obj)

this method returns true if specified element is available in the List otherwise it returns false (searching)

9.E get(int position)

this method returns the element from the List at specified position.

this method especially used to access the elements of the List.

10.E set(int position,E obj)

this method replace the new Element at specified position in the List.

```
//wap to demo on ArrayList
import java.util.*;
class ArrayListDemo{
public static void main(String args[]){
//creation of ArrayList and taking the
//generic parameter as Integer which means
//this collection will take only Integers
ArrayList<Integer> al = new ArrayList<Integer>();
//adding the elements into the List
al.add(10); //autoboxing
al.add(new Integer(20));//mannual boxing
al.add(30);
al.add(10);
al.add(40);
al.add(10);
al.add(50);
al.add(10);
//displaying elements of the List and its size
System.out.println(al);
System.out.println(al.size());
//inserting an element into the List at specified position
al.add(1,77);
System.out.println(al);
System.out.println(al.size());
//modifying the existing element of the List
al.set(4,al.get(2)+100);
System.out.println(al);
System.out.println(al.size());
//removing the element of the List by specifying its value
al.remove(new Integer(30));
System.out.println(al);
System.out.println(al.size());
//removing the element of the List by specifying its index position
al.remove(0);
System.out.println(al);
System.out.println(al.size());
//Displaying elements of List 1 by 1 using for loop(accessing)
for(int i=0;i<al.size();i++){
System.out.println(al.get(i));
```

```
}
//Displaying elements of List 1 by 1 using foreach loop
for(int v:al){  //autounboxing
 System.out.println(v);
 }
//searching an element
System.out.println(al.contains(50));
//copying the array List into another List
 ArrayList<Integer>  al1 = new ArrayList<Integer>(al);
 System.out.println(al1);
 }
}
```

Note:

1. In add(), remove(),get(),set() methods if we write any wrong index which is not available then it will throw a runtime exception called IndexOutOfBoundsException.
2. ArrayList supports to store multiple null values.

```
//wap to demo on ArrayList which stores different types of objets
import java.util.*;
class Student{
}
class  ArrayListDemo1{
   public static void main(String args[]){
 ArrayList al = new  ArrayList();
  al.add(10);
  al.add(20.0);
  al.add(20l);
  al.add(30f);
  al.add(null);
  al.add(true);
  al.add("abc");
  al.add('c');
  al.add(new Student());
  System.out.println(al);
 }
}
```

LinkedList

-LinkedList is the implementation class of List interface which is also used to store group of individual objects where duplicate values are allowed

-LinkedList internally follows doubly linked list structure where all the elements are stored in the form nodes which linked each other.

-LinkedList is not a synchronized class

-LinkedList is also suppert multiple null values

creation of LinkedList:

```
LinkedList<E>  ll = new  LinkedList<E>();
LinkedList<E>  ll = new  LinkedList<E>(Collection obj);
```

Note:

there is no capacity concept for LinkedList

Methods of LinkedList:

- we can use all Collection Methods to work with the Linked List

// wap to demo on LinkedList

IQ: what is the diff b/w ArrayList and LinkedList

ArrayList is slower in insertion and deletion of elements because it internally requires shifting operations, But faster in accessing the elements because ArrayList use index position for every element.

LinkedList is faster in insertion and deletion of elements because it just require modifying the links of nodes instead of shifting operations,But slower in accessing the elements because LinkedList does not use any index position.

Vector

-Vector is the implementation class of List interface which is also used to store group of individual objects where duplicate values are allowed

- Vector is exactly similar to ArrayList but ArrayList is not a synchronized class where Vector is a synchronized class
- Vector is also called legacy class because it is available from java 1.0 version.

creation of Vector:

```
Vector<E>  v = new Vector<E>();
Vector<E>  v = new Vector<E>(int capacity);
Vector<E>  v = new Vector<E>(Collection obj);
```

Methods

-we can use all collection Methods

-we can also use legacy methods like addElement(), removeElement(), setElementAt(),.....

```
//wap to demo on Vector
import java.util.*;
class  VectorDemo{
  public static void main(String args[]){
    Vector<String>  v =  new Vector<String>();
       v.add("sachin");
       v.add(new String("sehwagh"));
       v.add("kohli");
       v.add("dhoni");
       v.add("suresh");
  System.out.println(v);
  System.out.println(v.size());
  }
}
```

Stack

-Stack is a child class of Vector and implements List interface

-Stack stores a group of objects b using a mechanism called LIFO

-LIFO stands for Last in first out , it means last inserted element deleted first.

creation of Stack:

```
Stack<E>  s = new  Stack<E>();
```

Methods:

-We can use all collection Methods

-We can also use legacy methods of Vector class like addElement(), removeElement(), setElementAt(),.....

-But if we want to follow the LIFO mechanism we should use Stack methods like follows

1. E push(E obj)

this method will add new element into the Stack

2. E pop()

this method deletes the top element available on Stack

3. E peek()

this method just returns the top element available on Stack

```
// wap to demo on Stack
import  java.util.*;
class  StackDemo{
  static public void main(String args[]){
     Stack<Double>  s = new Stack<Double>();
        s.push(10.2);
        s.push(50.2);
        s.push(30.2);
        s.push(40.2);
        s.push(70.2);
    System.out.println(s);
    System.out.println(s.pop());
    System.out.println(s);
    System.out.println(s.peek());
    System.out.println(s);
  }
}
```

	ArrayList	LinkedList	Vector	Stack
ordered	ordered by insertion	ordered by insertion	ordered by insertion	ordered by insertion
data sstructure	growable array	doubly linked list	growable array	growable array
synchronized	✗	✗	✓	✓
duplicate values	✓	✓	✓	✓
null values	✓	✓	✓	✓
initialcapacity	10	NA	10	10
capacity increment	C*3/2+1	NA	2*C	2*C
legacy	✗	✗	✓	✓

cursors of Collection Framework

- cursors are mainly used to access the elements of any collection
- we have following 3 types of cursors in Collection Framework

1.Iterator
2.ListIterator
3.Enumeration

Iterator

- this cursor is used to access the elements in forward direction only
- this cursor can be applied Any Collection (List,Set)
- while accessing the methods we can also delete the elements
- Iterator is inteface and we can not create an object directly.

- if we want to create an object for Iterator we have to use iterator() method

creation of Iterator:

Iterator it = c.iterator();

here iterator() method internally creates and returns an object of a class which implements Iterator interface.

Methods

1. boolean hasNext()
2. Object next()
3. void remove()

//wap to demo on Iterator

```
import java.util.*;
class IteratorDemo{
 public static void main(String args[]){
    LinkedList<Integer> ll = new LinkedList<Integer>();
   ll.add(10);
   ll.add(25);
   ll.add(50);
   ll.add(20);
   ll.add(25);
   ll.add(23);
   ll.add(60);
   ll.add(25);
   ll.add(30);
   ll.add(40);
   ll.add(15);
   ll.add(25);
System.out.println(ll);
  Iterator it = ll.iterator();
while(it.hasNext()){
   System.out.println(it.next());
  }
    Iterator it1 = ll.iterator();
 while(it1.hasNext()){
  Integer e = (Integer) it1.next(); //down casting
  if(e==25){
    it1.remove();
  }
 }
  System.out.println(ll);
 }
}
```

//wap to remove second occurence of 25 in the above collection using Iterator

2. ListIterator

-this cursor is used to access the elements of Collection in both forward and backward directions

-this cursor can be applied only for List category Collections

-while accessing the methods we can also add,set,delete elements

-ListIterator is interface and we can not create object directly.

-if we want to create an object for ListIterator we have to use listIterator() method

creation of ListIterator:

ListIterator<E> it = l.listIterator();

here listIterator() method internally creates and returns an object of a class which implements ListIterator interface.

Methods

1. boolean hasNext();
2. Object next();
3. boolean hasPrevious();
4. Object previous();
5. int nextIndex();
6. int previousIndex();
7. void remove();
8. void set(Object obj);
9. void add(Object obj);

```
//wap to demo on ListIterator
import java.util.*;
class ListIteratorDemo{
 public static void main(String args[]){
 LinkedList<Integer> ll = new LinkedList<Integer>();
   ll.add(10);
   ll.add(25);
   ll.add(50);
   ll.add(20);
   ll.add(25);
   ll.add(30);
   ll.add(40);
   ll.add(15);
   ll.add(25);
  System.out.println(ll);
    ListIterator<Integer> lit = ll.listIterator();
  System.out.println("Elements in Forward Direction");
while(lit.hasNext()){
  System.out.print(lit.next()+"  ");
 }
  System.out.println("\nElements in Backward Direction");
while(lit.hasPrevious()){
  System.out.print(lit.previous()+"  ");
 }
while(lit.hasNext()){
      Object o = lit.next();
     Integer e = (Integer) o;
   if(e==23){
    // lit.add("sachin");
      lit.add(56);
   }
  if(e==15){
  lit.set(15000);
   }
  if(e==25){
  lit.remove();
  }}
```

```
   System.out.println("\nnew list:"+ll);
  }
}
//wap to print Elements in Backward Direction directly using
   ListIterator
```

3. Enumeration-

- this cursor is used to access the elements of Collection only in forward direction
- this is lagacy cursor can be applied only for legacy classes like Vector,Stack,Hashtable.

-Enumeration is also an interface and we can not create object directly.

-if we want to create an object for Enumeration we have to use a legacy method called elements() method

creation of Enumeration:

Enumeration e = v.elements();

here elements() method internally creates and returns an object of a class which implements Enumeration interface.

Methods

1. boolean hasMoreElements()
2. Object nextElement();

```
//wap to demo on Enumeration
import java.util.*;
class EnumerationDemo{
 public static void main(String args[]){
 Vector<Integer> v= new Vector<Integer>();
   v.add(10);
   v.add(25);
   v.add(50);
   v.add(20);
   v.add(25);
   v.add(23);
   v.add(25);
  System.out.println(v);
Enumeration e = v.elements();
while(e.hasMoreElements()){
  System.out.print(e.nextElement()+" ");
 }
}
}
```

	Iterator	ListIterator	Enumeration
Applicable to	List, Set	List implemented clases	legacy classes
Accessing Direction	only forward	both forward & backward	only forward
Operations	access, remove	access,remove, add, set	only accessing
method to create	iterator()	listIterator()	elements()
legacy	✗	✗	✓
No.of Methods	3	9	2

Set category

HashSet

-HashSet is the implementation class of Set interface which is also used to store group of individual objects but duplicate values are not allowed

-HashSet internally follows hashtable structure where all the elements are stored using hashing technique which will improve the performance by reducing the waiting time.
-HashSet is not a synchronized class
-HashSet supports only one null value.
-HashSet is called unordered Collection because it is not guarantee for insertion order of elements.

creation of HashSet:

```
HashSet<E> hs = new HashSet<E>();
HashSet<E> hs = new HashSet<E>(int capacity);
HashSet<E> hs = new HashSet<E>(int capacity,float loadfactor);
HashSet<E> hs = new HashSet<E>(Collection obj);
```

Methods

1. boolean add(E obj)
2. boolean remove(E obj)
3. int size()
4. void clear()
5. boolean contains(E obj)
6. boolean isEmpty()

//wap to demo on HashSet

```
import java.util.*;
class HashSetDemo{
  public static void main(String ar[]){
   HashSet<Integer> hs = new HashSet<Integer>();
        hs.add(17);
        hs.add(13);
        hs.add(27);
        hs.add(null);
        hs.add(12);
        hs.add(45);
 System.out.println(hs);
 System.out.println(hs.size());
    for(int e: hs){
     System.out.println(e);
    }
 Iterator it = hs.iterator();
  while(it.hasNext()){
  System.out.println(it.next());
 }
 } }
```

LinkedHashSet

-LinkedHashSet is the implementation class of Set interface which is also used to store group of individual objects but duplicate values are not allowed
-LinkedHashSet internally follows hashtable+doubly linked list structures
-LinkedHashSet is not a synchronized class
-LinkedHashSet supports only one null value.
-LinkedHashSet is called as ordered Collection because it is guarantee for insertion order of elements.

creation of LinkedHashSet:

```
LinkedHashSet<E> hs = new LinkedHashSet<E>();
LinkedHashSet<E> hs = new LinkedHashSet<E>(int capacity);
LinkedHashSet<E> hs = new LinkedHashSet<E>(int capacity,float loadfactor);
```

LinkedHashSet<E> hs = new LinkedHashSet<E>(Collection obj);
//wap to demo on LinkedHashSet

TreeSet

-TreeSet is the implementation class of Set interface which is also used to store group of individual objects but duplicate values are not allowed
-TreeSet internally follows tree structure
-TreeSet is not a synchronized class
-TreeSet is called as unordered Collection because it is not guarantee for insertion order of elements but all elements are stored in sorted order(bydefault ascending order)
- TreeSet supports only one null value if TreeSet is Empty otherwise TreeSet does not support null values because it internally perform comparision operations but we never compare a null value. with any object and it will throw a RuntimeException saying NullPointerException
-TreeSet does not allow to store different types of objects because it internally perform comparison.operations but we never compare a 2 different types of it will throw a RuntimeException saying ClassCastException

creation of TreeSet

TreeSet<E> ts = new TreeSet<E>();
TreeSet<E> ts = new TreeSet<E>(SortedSet);
TreeSet<E> ts = new TreeSet<E>(Comparator);
TreeSet<E> ts = new TreeSet<E>(Collection obj);
//wap to demo on TreeSet

	Hash Set	LInkedHashSet	TreeSet
ordered	unordered by insertion	ordered by insertion	unordered by insertion
data sstructure	Hashtable	Hashtable + DLL	tree structure
synchronized	X	X	X
duplicate values	X	X	X
null values	1 null	1 null	1null ifTreeSet empty otherwise no null value
initialcapacity	16	16	NA
capacity increment	2*C	2*C	NA
legacy	X	X	X

Collections

-Collections is a utilitiy class available in Collection FrameWork which contains some use full methods for manipulating elements of any collection like searching,sorting,finding min,max number...
//wap to deom on Collections class

```
import java.util.*;
class MyComparator implements Comparator{
  public int compare(Object o1,Object o2){
    Integer i1 = (Integer) o1;
    Integer i2 = (Integer) o2;
   if(i1>i2){
```

```
    return -1;
    }
   else
    if(i1<i2){
    return 1;
    }
   else{
    return 0;
    }
  }
}
class   CollectionsDemo{
 public static void main(String ar[]){
ArrayList<Integer> al = new ArrayList<Integer>();
   al.add(10);
   al.add(25);
   al.add(50);
   al.add(20);
   al.add(25);
   al.add(23);
   al.add(60);
   al.add(25);
   al.add(30);
   al.add(40);
   al.add(15);
   al.add(25);
System.out.println(al);
System.out.println(al.size());
Collections.reverse(al);
System.out.println(al);
Collections.sort(al);
System.out.println(al);
MyComparator mc = new MyComparator();
Collections.sort(al,mc);
System.out.println(al);
Collections.swap(al,0,1);
System.out.println(al);
System.out.println(Collections.min(al));
System.out.println(Collections.max(al));
Set<Integer> hs = new HashSet<Integer>(al);
System.out.println(hs);
System.out.println(hs.size());
/*
Collections.sort(hs); //Invalid
System.out.println(hs);
*/
hs =new TreeSet<Integer>(hs);
System.out.println(hs);
System.out.println(Collections.min(hs));
```

```
System.out.println(Collections.max(hs));
 }
}
```

Comparator:

Comparator is a interface using which we can decide our own sorted order. This comparator we use in 3 areas like inside the TreeSet, TreeMap, sort() method on Collections class.

```
//wap to maintain a group of elements in TreeSet in descending  order  using Comparator.
import  java.util.*;
class  MyComparator  implements  Comparator<String>{
  public   int  compare(String s1,String  s2){
   if(s1.compareTo(s2)>0){
    return -1;
    }
   else
   if(s1.compareTo(s2)<0){
    return 1;
    }
   else{
    return 0;
    }
  }
}
class    TreeSetDemo1{
 public  static  void  main(String ar[]){
TreeSet<String> ts=new TreeSet<String>(new MyComparator());
    ts.add("sachin");
    ts.add("dhoni");
    ts.add("kohli");
    ts.add("yuvaraj");
    ts.add("pujara");
    ts.add("rohit");
    ts.add("dhavan");
System.out.println(ts);
 }
}
```

Arrays:

Arrays is another utility class given java.util package which contains use full methods using which we an maipulate normal arrays like sorting, searching,....

```
//wap to demo on Arrays  class
import  java.util.*;
class    ArraysDemo{
 public  static  void  main(String ar[]){
 Integer  arr [] = {67,12,34,21,67,45,34,89,10};
System.out.println("\nArray Values Before Sorting....");
for(int v: arr){
System.out.print(v+"  ");
}
//Collections.sort(arr); - Invalid
  Arrays.sort(arr);
```

```
System.out.println("\nArray Values After Sorting....");
for(int v: arr){
System.out.print(v+" ");
}
//converting a normal array into Collection
List l = Arrays.asList(arr);
  // l.add(1000); - invalid
 ArrayList<Integer> al = new ArrayList<Integer>(l);
al.add(1000);
System.out.println("\nnew List:"+al);
 }
}
//WAP to convert collection into normal array
import java.util.*;
class   ToArrayDemo{
 public static void main(String ar[]){
ArrayList<Integer> al = new ArrayList<Integer>();
   al.add(10);
   al.add(25);
   al.add(50);
   al.add(20);
   al.add(25);
   al.add(23);
   al.add(60);
   al.add(25);
System.out.println(al);
//converting a Collection into normal array
Object arr[] = al.toArray();
System.out.println("\nArray Values....");
for(int i=0;i<arr.length;i++){
 System.out.print(arr[i]+" ");
}
}
}
//wap to copy list1,list2 into list3
import java.util.*;
class   CopyListDemo{
 public static void main(String ar[]){
ArrayList<Integer> al1 = new ArrayList<Integer>();
   al1.add(10);
   al1.add(25);
   al1.add(50);
   al1.add(20);
   al1.add(25);
System.out.println(al1);
ArrayList<Integer> al2 = new ArrayList<Integer>();
   al2.add(100);
   al2.add(250);
   al2.add(500);
```

```
    al2.add(200);
    al2.add(250);
System.out.println(al2);
ArrayList<Integer>  al3 = new  ArrayList<Integer>();
 al3.addAll(al1);
 al3.addAll(al2);
System.out.println(al3);
 }
}
```

Map category

HashMap

-HashMap is the implementation class of Map interface which is used to store group of objects in the form of Key-Value pairs where but Keys can not be duplicated but values can be duplicated

-HashMap internally follows hashtable data structure

-HashMap is not a synchronized class

-HashMap supports only one null value for Key Objects but we can store multiple null values for Value Object

-HashMap is called unordered Map because it is not guarantee for insertion order of elements.

creation of HashMap:

```
HashMap<K,V>  hm  = new HashMap<K,V>();
HashMap<K,V>  hm  = new HashMap<K,V>(int capacity);
HashMap<K,V>  hm  = new HashMap<K,V>(int capacity, float loadfactor);
HashMap<K,V>  hm  = new HashMap<K,V>(Map obj);
```

Methods of HashMap

1.V put(K obj,V obj)

this method is used to add new key-value pair into the Map

2.V get(K obj)

This method returns the Value object of the specified Key. If specified Key is not available then it returns null.

3.V remove(K obj)

This method removes the specified Key alongwith its value.

4.int size()

This method returns the count of the no.of Key-Value pairs available in the Map.

5.boolean isEmpty()

this Method returns true if Map not containing any Key-Value pairs otherwise it returns false.

6. void clear()

this Method clear all the Key-Value pairs of the Map.

7.Set keySet()

this method returns all the keys available in the Map separately in the form of Set.

8.Collection values()

this method returns all the values available in the Map separately in the form of Collection.

9.Set entrySet()

this method returns all the key-values or entries available in the Map separately in the form of Set.

10. boolean containsKey(K obj)

this method returns true if specified Key is available in the Map otherwise it returns false.

11. boolean containsValue(V obj)

this method returns true if specified Value is available in the Map otherwise it returns false.

```
//wap to demo on HashMap
import java.util.*;
class  HashMapDemo{
 public  static  void  main(String  args[]){
```

```
HashMap<Integer,String>hm=newHashMap<Integer,String>();
  hm.put(11,"sachin");
  hm.put(37,"dhoni");
  hm.put(25,"kohli");
  hm.put(13,"raina");
  hm.put(12,"yuvaraj");
System.out.println(hm);
System.out.println(hm.size());
Set ks = hm.keySet();
System.out.println(ks);
Collection cv = hm.values();
System.out.println(cv);
Set entry = hm.entrySet();
System.out.println(entry);
System.out.println(hm.containsKey(12));
System.out.println(hm.remove(25));
System.out.println(hm);
  }
}
```

LinkedHashMap

-LinkedHashMap is the implementation class of Map interface which is also usedto store group of3 objects in the form of Key-Value pairs where Keys can't be duplicated but values can be duplicated

-LinkedHashMap internally follows hashtable+doubly linked list structures

-LinkedHashMap is not a synchronized class

-LinkedHashMap supports only one null value for Key Objects but we can store multiple null values for Value Object

-LinkedHashMap is called as ordered Map because it is guarantee for insertion order of elements.

TreeMap

-TreeMap is the implementation class of Map interface which is also used to store group of objects in the form of Key-Value pairs where Keys can't be duplicated but values can be duplicated.

-TreeMap internally follows tree structure

-TreeMap is not a synchronized class

-TreeMap is called as unordered Map because it is not guarantee for insertion order of elements, but. all elements are stored in sorted order(bydefault ascending order based on keys)

-TreeMap does not supports null values for Key Objects but we can store multiple null values for Value Objects

Hashtable

-Hashtable is the implementation class of Map interface which is also used to store group of objects in the form of Key-Value pairs where Keys can't be duplicated but values can be duplicated

-Hashtable is exactly similar to HasMap but Hashtable is a synchronized class where HashMap is not a synchronized class

-Hashtable does not support null values for both Keys and Values

```
//wap to demo on Hashtable
import java.util.*;
class HashtableDemo{
 public static void main(String args[]){
Hashtable<String,Integer> ht = new Hashtable<String,Integer>();
      ht.put("sachin",200);
      ht.put("rohit",264);
```

```
        ht.put("sehwagh",219);
        ht.put("ganguly",183);
        ht.put("dhoni",183);
      System.out.println(ht);

Enumeration e = ht.keys();
while(e.hasMoreElements()){
System.out.println(e.nextElement());
}
}
}
```

	HashMap	LinkedHashMap	TreeMap	Hashtable
ordered	unordered by insertion	ordered by insertion	unordered by insertion	unordered by insertion
data sstructure	Hashtable	Hashtable + DLL	tree structure	Hashtable
synchronized	✗	✗	✗	✓
duplicate values	K✗ V✓	K✗ V✓	K✗ V✓	K✗ V✓
null values	1 null for Keys n null for Values	1 null for Keys n null for Values	0 null for Keys n null for Values	0K, 0V
initialcapacity	16	16	NA	11
capacity increment	2*C	2*C	NA	2*C
legacy	✗	✗	✗	✓

? Can we stop null values into ArrayList

yes we can stop null values into ArrayList for achieving this requirement we have to use Collections.checkedList() method

Eg:

```
import java.util.*;
class Test1{
  public static void main(String args[]){
   List<Integer> al = new ArrayList<Integer>();
 // al = Collections.checkedList(al,Integer.class);
   al.add(20);
  al.add(30);
  al.add(null);
  al.add(30);
  al.add(20);
  al.add(null);
  al.add(50);
  al.add(30);
```

```
   System.out.println(al);
   System.out.println(al.size());
  }
}
```

In the above program if we remove the comments symbols then List does not support any null value and it returns NullPointerException at runtime,but if we comment the particular line again it will accept null values.

similarly we can use Collections.checkedSet(), Collections.checkedMap() methods to stop null values into Set and Map categories.

?can we convert ArrayList object as synchronized object

yes we can convert ArrayList object as synchronized object for achieving this requirement we have to use Collections.synchronizedList() method.

Eg:

```
List<Integer> al = new ArrayList<Integer>()
 al.add(10);
 al.add(20);
 al.add(30);
..
-- Now here  al is the not synchronized object.
al  = Collections.synchronizedList(al);
-  Now here  al is a synchronized object.
```

- similarly we can use Collections.synchronizedSet(), Collections.synchronizedMap() methods to make Set and Map objects as synchronized.

```
//wap  to store student objects into Collection
import  java.util.*;
class  Student{
int rno;
String name;
Student(int rno,String name){
this.rno=rno;
this.name=name;
}
void display(){
System.out.println(rno+"\t"+name);
}
public  String  toString(){
 return rno+"="+name;
}
}
class  StudentCollection{
 public static void main(String args[]){
    HashSet<Student>  hs = new HashSet<Student>();
/*
 //case-1
     hs.add(new Student(1,"sachin"));
     hs.add(new Student(1,"sachin"));
     hs.add(new Student(1,"sachin"));
     hs.add(new Student(1,"sachin"));
     hs.add(new Student(1,"sachin"));
```

```
      hs.add(new Student(1,"sachin"));
    System.out.println(hs);
     System.out.println(hs.size());//6
    */
/*
  //case-2
  Student s = new    Student(1,"sachin");
  hs.add(s);
  hs.add(s);
  hs.add(s);
  hs.add(s);
  hs.add(s);
System.out.println(hs);
System.out.println(hs.size());//1
*/
      hs.add(new Student(1,"sachin"));
      hs.add(new Student(2,"sachin"));
      hs.add(new Student(3,"kohli"));
      hs.add(new Student(4,"dhoni"));
      hs.add(new Student(5,"suresh"));
      hs.add(new Student(6,"yuvaraj"));
     System.out.println(hs);
     System.out.println(hs.size());
    Iterator  it = hs.iterator();
while(it.hasNext()){
   Student ss = (Student) it.next();
    ss.display();
   if(ss.rno==3){
     it.remove();
  }
 }
 System.out.println(hs);
 }
}
```

Note:

1.In case1 6 different objects are stored into HashSet which containing same data sothat the size of the Hashset is 6.

2.In case2 same oject is stored into HashSet for6 times but HashSet does not support duplicate values sothat the size of the Hashset is 1.

3.When we display HashSet using System.out.println() it will display hashcode of each object but if we want to displayobject data instead of hashcode we need to override toString()

//wap to store student objects into TreeSet

StringTokenizer

StringTokenizer is utitility class available in java util package which is used to break the String into multiple pieces or tokens

creation of StringTokenizer

StringTokenizer st = new StringTokenizer("string")

here it will take the default delimeter as separator between the tokens i.e, space.

StringTokenizer st=new StringTokenizer("string","delemeter")

here it will take the specified delimeter as separator between the tokens.

```
//wap to demo on StringTokenizer
import java.util.*;
class StringTokenizerTest{
public static void main(String ar[]){
  String str = "Happy,New,Year.every;body,welcome;to.2015";
StringTokenizer st = new StringTokenizer(str,",;.");
System.out.println("No.of Words or Tokens"+st.countTokens());
while(st.hasMoreTokens()){
System.out.println(st.nextToken().toUpperCase());
}
 }
}
```

Calendar

- Calendar used to get the System Date and time in to the program
- Calendar is a abstract class we can not create any object directly but if we want to create anyobject for Calendar then we have to use a factory method called getInstance();

creation of Calendar:

```
    Calendar c = Calendar.getInstance();
//wap to demo on Calendar
import java.util.*;
class CalendarDemo{
public static void main(String args[]){
String months[]={"JAN","FEB","MAR","APR","MAY","JUN","JUL", "AUG","SEP", "OCT","NOV","DEC"};
/*
 Date d = new Date();
 System.out.println(d);
 System.out.println(d.getYear()); */
Calendar c = Calendar.getInstance();
int y = c.get(Calendar.YEAR);
int m =c.get(Calendar.MONTH);
int da = c.get(Calendar.DATE);
System.out.println(da+"-"+months[m]+"-"+y);
int h = c.get(Calendar.HOUR);
int mi = c.get(Calendar.MINUTE);
int s = c.get(Calendar.SECOND);
int amorpm = c.get(Calendar.AM_PM);
if(amorpm==0){
System.out.println(h+":"+mi+":"+s+" AM");
}
else{
System.out.println(h+":"+mi+":"+s+" PM");
```

```
}
System.out.println(c.get(Calendar.DAY_OF_YEAR));
}
}
```

Generics

generic programming

generic programming means reusing the same code for storing different types of objects.

observe the following program

```
class Demo{
Integer e;
void add(Integer e){
this.e=e;
}
Integer getValue(){
return e;
}
}
class GPTest1{
public static void main(String args[]){
Demo d = new Demo();
d.add(10);
Integer i = d.getValue();
System.out.println("i="+i);
/*
d.add("abc");
String str = d.getValue();
System.out.println("str="+str);
d.add(10.2);
Double dd = d.getValue();
System.out.println("dd="+dd); */
}}
```

Note:

-In the above program class Demo can hold only Integer objects but it can not hold any other objects like Float objects, String objects,....

-It means we have typesafty for the data and while retrieving typecasting is optional but there is no generic programming approach.

-To resolve this problem we have to create many classes to store the different objects which will increase the code size and maintenance becomes complicated.

To achieve generic programming in our application we have to use java's super most class called Object(upto java1.5)

observe the following program

```
class Demo{
Object e;
void add(Object e){
this.e=e;
}
Object getValue(){
return e;
}}
class GPTest2{
```

```
public static void main(String args[]){
Demo d = new Demo();
d.add(10);
Integer i = (Integer)d.getValue();
System.out.println("i="+i);
d.add("abc");
String str = (String)d.getValue();
System.out.println("str="+str);
d.add(10.2);
Double dd = (Double) d.getValue();
System.out.println("dd="+dd);
}
}
```

Note:

-In the above program class Demo can hold any kind of object like Integer objects, Float objects, String objects,....

-It means here we have generic programming approach.

-But in this approach we have following 2 drawbacks

1.there is no type safty for the data. for example if we want to store salary values we should enter all the integer values and it will accept but in the middle unexpectedly we enter any wrong value like string or double or other objects still it will accept, so that there is no type safty.

2. while retrieving the data type casting is mandatory.

To achieve generic programming and resolving the problems of no type safty and need for type casting java people have introduced a concept called generics.

observe the following program

```
class Demo<T>{
T e;
void add(T e){
this.e=e;
}
T getValue(){
return e;
}
}
class GPTest3{
public static void main(String args[]){
//case-1
Demo<Integer> d1 = new Demo<Integer>();
d1.add(10);
Integer i = d1.getValue();
System.out.println("i="+i);
//d1.add("abc"); - invalid it wont accept
//case-2
Demo<String> d2 = new Demo<String>();
d2.add("abc");
String str =d2.getValue();
System.out.println("str="+str);
//case-3
Demo d3 = new Demo();
```

```
d3.add(10);
Integer  i1 = (Integer)  d3.getValue();
System.out.println("i1="+i1);
d3.add("abc");
String str1 =(String)  d3.getValue();
System.out.println("str1="+str1);
}
}
```

Note:

-In the above program class Demo can store any kind of object it means here we have generic programming approach.
-In case-1 object d1 is created by specifying generic parameter as Integer so object d1 can hold only Integer objects (type safty) and while retrieving the data typecasting is optional
-In case-2 object d2 is created by specifying generic parameter as String so object d2 can hold only String objects(type safty) and while retrieving the data typecasting is optional
-But In case-3 object d3 is created without specifying any generic parameter type so object d3 can hold any kind of objects (no type safty) and while retrieving the data typecasting is mandatory.

Generics

-Generics concept introduced in java1.5 version which is used to achieve generic programming and resolving the problems of type safty and need for type casting
-Generics can also be called as generic parameter types
-Generics concept can be used only for storing objects but not for primitive values.
-Generics can be called as type erasers because the generic information is available only upto compilation,once compilation is done then all the generic information will be erased.
-Using generics concept we can achieve compile time polymorphism
-This generic concept looks like templates concept in C++
-We can apply generics concept for a classes,interfaces and for methods.

Generic class

Generic class is a class which can hold any kind of objects, to create Generic class we have to specify generic type <T> after the class name.

syntax:

```
class  ClassName<T>{
    //members
 }
```

Note:

We can specify n number of generic types with a class like follows,

```
class ClassName<T1,T2,T3,...>{
    }
```

Generic interfaces

we can also create a generic interface by specifying the <T> after the interface name.

syntax:

```
interface  InterfaceName<T>{
   //members
  }
```

```
//wap to demo on Gneric Interfaces
interface  Tester<T>{
 void show(T o);
}

//implementing the generic interface without specifying
```

```
//any generic parameter
class Imp1 implements Tester{
 public void show(Object o){
  }
 }
//implementing the generic interface by specifying
//generic parameter as Integer
class Imp2 implements Tester<Integer>{
 public void show(Integer o){
 }
}
//implementing the generic interface by specifying
//generic parameter as String
class Imp3 implements Tester<String>{
public void show(String o){
}
}
//implementing the generic interface whose implementation
//class is also a generic class
class Imp4<T> implements Tester<T>{
public void show(T o){
}
}
class GenericInterface{
public static void main(String args[]){
 Imp1 i1 = new Imp1();
 i1.show(10);
 i1.show("abc");
 i1.show(true);
 Imp2 i2 = new Imp2();
 i2.show(10);
//i2.show("abc"); -invalid
 Imp3 i3 = new Imp3();
 i3.show("abc");
//i3.show(10.0); -invalid
Imp4<Integer> i4 = new Imp4<integer>();
 i4.show(10);
 }
}
```

Generic Method

-Generic method is a method which can take any kind of parameter.

-To create the generic method we have to specify the generic type <T> before the return of the method.

syntax:

```
 <T> returntype methodName(parameters){
  //statements
  }
//Wap to Demo on GenericMethod
class Demo{
<T> void show(T o){
```

```
System.out.println("o="+o);
}
}
class GenericMethod{
public static void main(String args[]){
 Demo d = new Demo();
 d.show(10);
 d.show(10.0);
 d.show("abc");
 d.show('c');
 d.show(true);
 d.show(10f);
 d.show(10L);
 }
}
```

Comparable:

- Comparable is a interface available in java.lang package which is used to make particular class objects as comparable objects so that we can store into any TreeSet or TreeMap.
- If our class is not implementing this itnerface then we can't store our class objects into TreeSet or TreeMap and it will throw a runtime exception saying ClassCastException

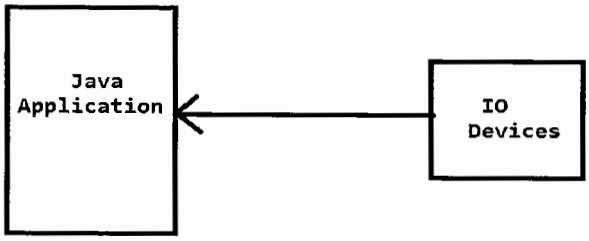
```
//wap to demo on Comparable
import java.util.*;
class Student implements Comparable<Student>{
int rno;
String name;
Student(int rno,String name){
 this.rno=rno;
 this.name=name;
}
void display(){
 System.out.println(rno+"\t"+name);
}
public String toString(){
return rno+"="+name;
}
public int compareTo(Student s){
 if(this.rno>s.rno){
   return 1;
  }
  else
  if(this.rno<s.rno){
   return -1;
  }
  else{
   return 0;
}
}
}
class StudentCollection{
```

```
public static void main(String args[]){
  TreeSet<Student>  hs = new TreeSet<Student>();
  hs.add(new Student(1,"sachin"));
  hs.add(new Student(2,"kohli"));
  hs.add(new Student(3,"dhoni"));
  hs.add(new Student(4,"yuvaraj"));
  hs.add(new Student(5,"sehwagh"));
  System.out.println(hs);
 }
}
```

IO Streams

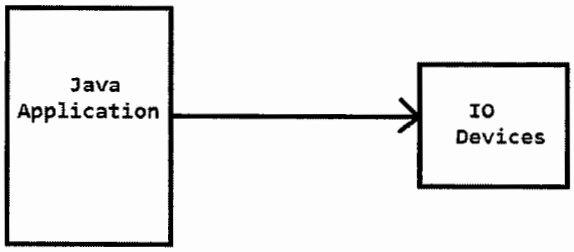
I- Input

Input means taking the data into out application which is nothing but reading data.



O- Output

Output means giving the data out from our application which is nothing but writing data.



Streams

Stream means flow of data from 1 location to another location.

IO Streams

- IO Streams are mainly used to transfer the data from one location to another location.
- IO Streams meant for performing reading operation from or writing operation onto any IO devices
- IO Streams are classified following 2 types

1. Byte Streams

- Byte Streams used to transfer the data from one location to another location byte by byte
- Byte Streams can handle any kind of Files like images, audio files, text files,....
- Byte Streams are again classified into following 2 types

1. InputStream

- InputStream is super most class of all classes which are used for performing reading operation from any input device byte by byte

Eg:

DataInputStream, FileInputStream,....

2. OutputStream

- OutputStream is super most class of all classes which are used for performing writing operation from any output

device byte by byte

Eg:

DataOutputStream,FileOutputStream,PrintStream....

2. Character Streams

-Character Streams are used to transfer the data from one location to another location char by char

-Byte Streams can handle only text files hence these streams can also be called as text streams

-Character Streams are faster than Byte Streams

- Character Streams are also classified into following 2 types

1.Reader

- Reader is super most class of all classes which are used for performing reading operation from any input device char by char

Eg:

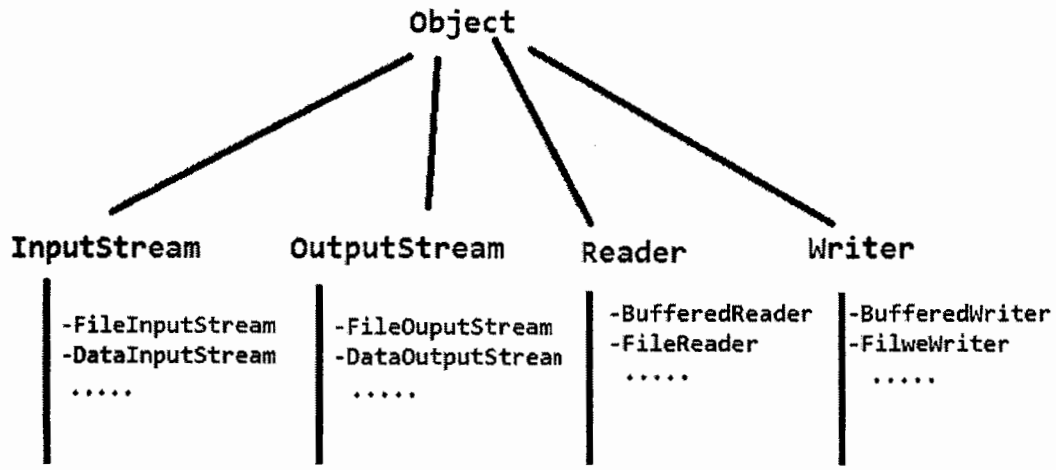
BufferedReader, FileReader,...

2.Writer

- Writer is super most class of all classes which are used for performing writing operation from any output device char by char

Eg:

BufferedWriter,FileWriter,PrintWriter...



DataInputStream

DataInputStream is a byte stream class which is usedto perform reading operation from any input device like keyboard,file,other machine,...

creation of DataInputStream:

DataInputStream dis = new DataInputStream(resource);

```
//wap to read the data fro keyboard using DataInputStream
import java.io.*;
class DISDemo{
   public static void main(String ar[]) throws IOException{
  DataInputStream dis = new DataInputStream(System.in);
 System.out.println("Enter Any String");
  String str = dis.readLine();
 System.out.println("str="+str);
 System.out.println("Enter Any Integer");
  int i = Integer.parseInt(dis.readLine()); //parsing
 System.out.println("i="+i);
 System.out.println("Enter Any Double");
  double d = Double.parseDouble(dis.readLine());
```

```
 System.out.println("d="+d);
 System.out.println("Enter Any Character");
  char  ch = (char)  dis.read(); //type casting
  System.out.println("ch="+ch);
  }
}
```

ByteArrayInputStream

ByteArrayInputStream is a byte stream class which is used to read the bytes of byte array byte by byte.

creation of ByteArrayInputStream:

```
  ByteArrayInputStream  bais=new ByteArrayInputStream(byte[]);
//wap to demo on   ByteArrayInputStream
import  java.io.*;
class  BAISDemo{
    public static void main(String[]   args){
// byte  arrb[] = {56,78,89,46,34,45,66};
String str = "Im a great Person in the World but this is my openion only";
  byte  arrb[] = str.getBytes();
ByteArrayInputStream  bais = new ByteArrayInputStream(arrb);
 int  ch;
   ch =  bais.read();
 while(ch != -1){
 System.out.print((char) ch);
   ch =  bais.read();
    }
 }
}
```

Note:

here read() method returns -1 if there are no bytes to read which means it reaches to end position

Files

- A File is a collection of information where data is available permanently.
- To Work withe files we have to use Following Streams
  - FileInputStream
  - FileOutputStream
  - FileReader
  - FileWriter
  - File

FileInputStream

this stream is used to read the content of the specified file byte by byte.

creation of FileInputStream:

```
  FileInputStream   fis = new   FileInputStream("filename");
// wap to read the data of a File
import  java.io.*;
class  FileReading1{
 public static void main(String   args[]) throws IOException{
  FileInputStream  fis = new FileInputStream ("input.txt");
   int ch;
 ch = fis.read();
while(ch != -1){
  System.out.print((char)  ch);
```

```
  ch = fis.read();
   }
fis.close();
//System.in.read();
 }
}
```

Note:

-While reading the data from a file if specified file is not available then it will throw a run time exception called FileNotFoundException.

-fis.read() method returns -1 if there are no bytes to read which is nothing but end position of the file.

BufferedInputStream

In the above program we are getting every char from the hard disk one by one so that it will take more time for reading the data. To resolve the problem and improve the performance we can use BufferedInputStream class using which we can fetch all file data at a time into our program it means we no need to go to hard disk several times.

```
//wap to read the file using BufferedInputStream
import java.io.*;
class FileReading11{
public static void main(String args[]) throws IOException{
FileInputStream fis = new FileInputStream("class.txt");
BufferedInputStream bis = new BufferedInputStream(fis);
int ch;
long start = System.currentTimeMillis();
ch = bis.read();
while(ch!=-1){
System.out.print((char) ch);
ch = bis.read();
}
long end = System.currentTimeMillis();
System.out.print("\nTime Taken:"+(end-start));
bis.close();
}
}
```

FileOutputStream

this byte stream is used to write the content on to a file byte by byte.

creation of FileOutputStream:

syntax1:

```
FileOutputStream fos = new FileOutputStream("filename");
```

this syntax will create the file if the specified file is not available and write the content otherwise if file is available then it will overwrite the old content with new content

syntax2:

```
FileOutputStream fos=new FileOutputStream("filename",true);
```

this syntax will create the file if the specified file is not available and write the content otherwise if file is available then it will append old content with new content

```
//Wap to write the content on to the file
import java.io.*;
class FileWriting1{
 public static void main(String args[]) throws IOException{
  FileOutputStream fos=newFileOutputStream("output.txt",true);
```

```
   DataInputStream  dis  = new  DataInputStream(System.in);
  char  ch;
 System.out.println("Enter chars and press # to save");
    ch = (char)  dis.read();
  while (ch != '#'){   fos.write(ch);
   ch = (char)  dis.read();
    }
 //releasing the resources
  fis.close();
  dis.close();
System.out.println("File is Created Successfully...");
}
}
```

Note:

- While writing data on to a file if specified file is not available then it wont throw any run time exception saying FileNotFoundException instead of this it will create the file with the specified name.
- But It will throw FileNotFoundException if the specified location is not available

```
//wap to copy the content of one file into another file
import java.io.*;
class   FileCopy{
  public  static  void  main(String  args[]) throws IOException{
    FileInputStream  fis = new FileInputStream(args[0]);
 FileOutputStream fos = new FileOutputStream(args[1]);
  int ch;
     ch = fis.read();
 while( ch != -1){
     fos.write(ch);
     ch =fis.read();
  }
fis.close();
fos.close();
System.out.println("\nFile Copied Successfully...");
  }
}
//wap to copy the content of one file into another file 2 times
//wap to copy the content of file1, file2 into file3
```

FileReader

this is character stream class which is used read the file information char by char.

creation of FileReader:

```
   FileReader  fr  = new   FileReader("filename");
//wap to demo on FileReader(read the file line by line)
import java.io.*;
class   FileReading2{
public static void main(String args[]) throws IOException{
FileReader fr = new FileReader("input.txt");
BufferedReader   br = new BufferedReader(fr);
String str;
str =  br.readLine();
int c=0;
```

```
while(str != null){
c++;
System.out.println(str);
str = br.readLine();
}
br.close();
System.out.println("c="+c);
}
}
//wap read the file line by line using DataInputStream
```

FileWriter

this character stream class used to write the content on to the file char by char.

creation of FileWriter:

```
FileWriter fw = new FileWriter("filename"); //overwrite
FileWriter fw = new FileWriter("filename",true); //append
// wap to demo on FileWriter
import java.io.*;
class FileWriting2{
public static void main(String args[]) throws IOException{
FileWriter fw = new FileWriter("output1.txt");
//writing the string on to the file
String str="this time never come back";
fw.write(str);
//writing the char array on to the file
char chars[] = str.toCharArray();
fw.write(chars);
//writing char by char on to the file
for(int i=0;i<str.length();i++){
fw.write(str.charAt(i));
}
//flushing or clearing the data from buffer
//and writing on to the file
fw.flush();
fw.close();
}
}
```

File:

-This class is used get the properties of a file like what is the length of the file,what is the path of file,whether it is file or not, readable or not, writable or not, existed or not,.....

- using File class we can also delete the file, rename file
- File class is not meant for reading the file or writing the file

creation of File:

```
File f = new File("filename");
//wap to demo on File
import java.io.*;
class FileDemo{
public static void main(String args[]) throws IOException{
File f = new File(args[0]);
if(f.exists()){
```

```
System.out.println("Existed:"+f.exists());
System.out.println("File:"+f.isFile());
System.out.println("Directory:"+f.isDirectory());
System.out.println("Readable:"+f.canRead());
System.out.println("Writable:"+f.canWrite());
System.out.println("Path:"+f.getPath());
System.out.println("Absolute Path:"+f.getAbsolutePath());
System.out.println("size:"+f.length()/1024f+"KB");
}
else{
System.out.println("Sorry File is not available");
}
}
}
```

BufferedReader

This is a character stream class which is used to perform reading operation from any input devicelike keyborad, file...

creation of BufferedReader:

```
BufferedReader br = new BufferedReader(Reader obj);
```

```
//wap to demo on BufferedReader
import java.io.*;
class BRDemo{
public static void main(String args[]) throws IOException{
InputStreamReader isr = new InputStreamReader(System.in);
BufferedReader br = new BufferedReader(isr);
}
}
```

Note:

To connect the character stream classes with byte stream classes we have to use InputStreamReader or OutputStreamWriter classes.

System class

-System class is a predefine class contains some usefull methods like currentTimeMillis(), gc(), exit(),setIn(),setOut(), setErr(), setProperty(),getProperty().....

- and also contains following 3 reference variables

1. out

out is the reference variable of PrintStream class holding object of PrintStream class which is by default connected to standard output device called Monitor.

2. err

err is also reference variable of PrintStream class holding object of PrintStream class which is by default connected to standard output device called Monitor as well.

3. in

in is the reference variable of InputStream class holding object of PrintStream class which is by default connected to standard input device called keyboard.

```
//wap to read char from keyboard without using any stream
import java.io.*;
class ReadChar1{
 public static void main(String ar[]) throws IOException{
   System.out.println("Enter Any Character");
   char ch = (char) System.in.read();
   System.out.println("ch="+ch);
```

```
  }
 }
//wap to read multiple chars from keyboard without using any stream
import  java.io.*;
class  ReadChar2{
   public   static  void  main(String ar[]) throws IOException{
 System.out.println("Enter multiple Chars and press # to stop");
   char  ch;
String str="";
while((ch = (char)  System.in.read()) != '#'){
  str = str+ch;
   }
  System.out.println("str="+str);
 }
}
//wap to write the contents on to a file using System.out
import java.io.*;
class   Divert1{
  public  static  void  main(String  args[]) throws IOException{
  FileOutputStream fos = new FileOutputStream("divert.txt");
PrintStream  ps1 = new PrintStream(fos);
PrintStream  ps2 = new PrintStream(System.out);
//diverting System.out  from  Monitor to file
  System.setOut(ps1);
System.out.println("Hi");
System.out.println("How Are you");
System.out.println("Are U Practicing java");
System.out.println("Attend For interviews");
System.out.println("get The JOb");
System.out.println("All the Best");
//re-diverting System.out  from  file to Monitor
  System.setOut(ps2);
System.out.println("File is created successfully.....");
  } }
//wap to read the file using   System.in
import  java.io.*;
class  Divert2{
   public  static  void main(String...  args) throws IOException{
    FileInputStream  fis = new FileInputStream("divert.txt");
        System.setIn(fis);
         int ch;
      while((ch = System.in.read()) != -1){
            System.out.print((char) ch);
          }
        fis.close();
    }}
```

Deflater Streams

- Deflater Streams are used to compress the data of a file once the data is compressed we can not read the compressed data directly sothat we can provide security for our data.

- we compress the data of the file either reading or writing a File
- we have following 2 types of Deflater Streams

1. DeflaterInputStream

this stream is used to compress the data of the file while reading a file.

creation of DeflaterInputStream:

DeflaterInputStream dis = new DeflaterInputStream(InputStream);

2. DeflaterOutputStream

this stream is used to compress the data of the file while writing a file.

creation of DeflaterOutputStream:

DeflaterOutputStream dos=new DeflaterOutputStream (OutputStream);

Inflater Streams

-Inflater Streams are used to uncompress the data of the compressed file. we can not read a compressed file directly we have to use in flater streams to un compress the file so that we can read a file.

-we can also uncompress the file either reading a file or writing a file

-we have following 2 types of InflaterSreams

1. InflaterInputStream

- this stream is used to uncompress the data of the file while reading a file.

creation of InflaterInputStream:

InflaterInputStream iis=new InflaterInputStream(InputStream)

2. InflaterOutputStream

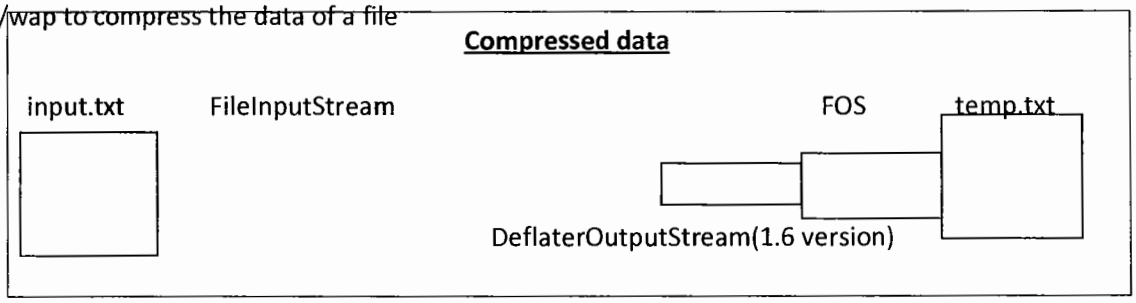
- this stream is used to uncompress the data of the file while writing a file.

creation of InflaterOutputStream:

InflaterOutputStream ios=new InflaterOutputStream(OutputStream)

all these Deflater and Inflater Stream classes are introduced in java 1.6 version and available in java.util.zip package.

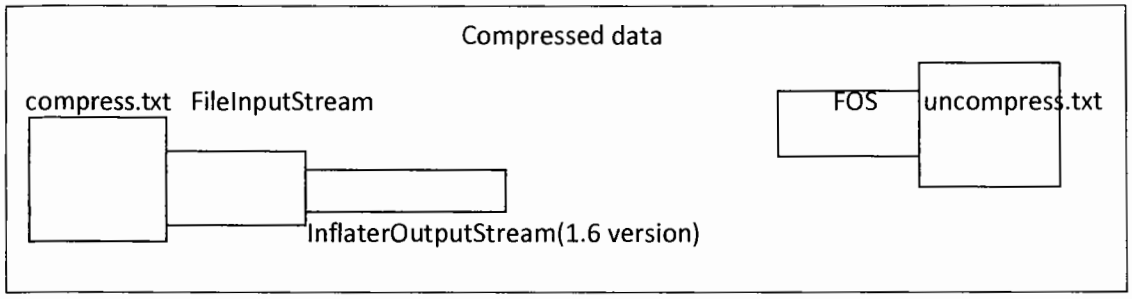
//wap to compress the data of a file




```
import java.io.*;
import java.util.zip.*;
class  CompressData{
public static void  main(String args[]) throws IOException {
 FileInputStream fis = new FileInputStream("input.txt");
 FileOutputStream fos = new FileOutputStream("compress.txt");
DeflaterOutputStream dos = new DeflaterOutputStream(fis);
int ch;
ch=fis.read();
while(ch!=-1){
//System.out.println((char)ch);
dos.write(ch);
ch=fis.read();
}
```

```
System.out.println("File is Compressed.");
fis.close();
dis.close();
}
}
```

//wap to uncompress the data of compressed file




```
import java.io.*;
import java.util.zip.*;
class UnCompressData{
public static void main(String args[]) throws IOException {
 FileInputStream fis = new FileInputStream("compress.txt");
 FileOutputStream fos = new FileOutputStream("uncompress.txt");
 InflaterInputStream iis = new InflaterInputStream(fis);
int ch;
ch=iis.read();
while(ch!=-1){
System.out.println((char)ch);
fos.write(ch);
ch=iis.read();
}
System.out.println("File is UnCompressed.");
}
}
```

Note:

-while uncompressing the file input file must be a compressed file otherwise it throws a run time Exception saying ZIPException.

-but while compressing file input file can be anything it may be normal file or compressed file.

Scanner

-Scanner is utility class introduced in java 1.5 version.

-Scanner is used t perform reading operation from any device like keyboard, file, ...

- Scanner available in java.util package but used in IO Streams

creation of Scanner:

Scanner sc = new Scanner(resource)

// wap to read the data from a Keyboard using Scanner

```
import java.util.*;
class Scanner1{
public static void main(String args[]){
  Scanner sc = new Scanner(System.in);
 System.err.println("Enter Any String");
  String str=sc.next();
```

```
  System.err.println("str="+str);
  System.err.println("Enter Any Integer");
  int i= sc.nextInt();
  System.err.println("i="+i);
  System.err.println("Enter Any Double");
  double d=sc.nextDouble();
  System.err.println("d="+d);
  System.err.println("Enter Any Character");
  char ch= sc.next().charAt(0);
  System.err.println("ch="+ch);
 }
}
//wap to read a file using Scanner
import java.util.*;
import java.io.*;
class Scanner2{
public static void main(String args[]) throws IOException{
FileInputStream fis = new FileInputStream("input.txt");
Scanner sc = new Scanner(fis);
/*
//reading the file word by word
while(sc.hasNext()){
 System.out.println(sc.next());
}
*/
//reading the file line by line
while(sc.hasNextLine()){
System.out.println(sc.nextLine());
}
sc.close();
}
}
//wap to break the String using Scanner
import java.util.*;
class Scanner3{
public static void main(String args[]){
String str = "this,is,a,long,string,because,it,contains,many chars";
Scanner sc = new Scanner(str);
sc.useDelimiter(",");
while(sc.hasNext()){
System.out.println(sc.next());
}
}
}
```

Object files

- Object file is a collection of different types of objects

-to create the object files we have to take the help of streams like ObjectOutputStream , ObjectInputStream

-to write the object or read the object then particular object must be serialized
-an object can said to be serialized only when its class implementing Serializable interface
-if we are reading or writitng any object which is not serialized then jvm will throw a run time error or Exception saying NotSerializableException

Serializable interface

-Serializable is a interface defined in the java.io package with 0 methods.
-if any interface is defined with 0 methods then it is called as marked interface or tagged interface
-the main job of marked or tagged interface is providing instructions to jvm to perform special task.

Eg:

Cloneable, Serializable, EventListener,....

-if we are declaring a class by implementing a serializable interface then we are giving an instruction to the jvm to allow us to read the object from, write the object into object files.

Serialization:

serialization is a process of converting the object into stream of bytes which is nothing but writing the object in to object file. To perform this serialization we use a class called ObjectOutputStream.

creation of ObjectOutputStream:

ObjectOutputStream oos=new ObjectOutputStream (OutputStream);

de-serialization:

de-serialization is a process of converting stream of bytes into object which is nothing but reading the object from object file. to perform this de-serialization we use the class called ObjectInputStream.

creation of ObjectInputStream:

ObjectInputStream ois=new ObjectInputStream(InputStream);

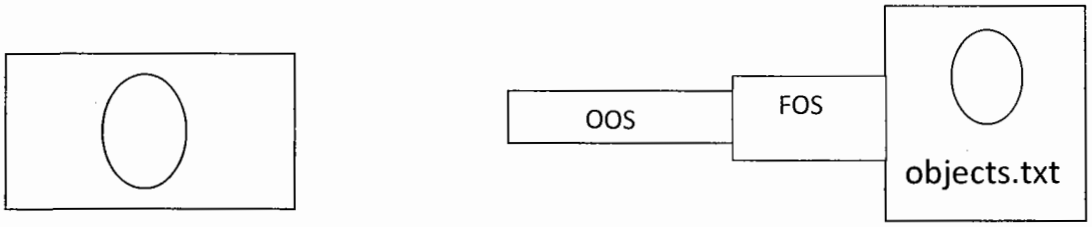
progrm to demo on creating and reading object files

Student.java

```
import java.io.Serializable;
class Student implements Serializable{
 int rno;
 String name;
 String address;
Student(int rno,String name,String address){
 this.rno=rno;
 this.name=name;
 this.address=address;
 }
void display(){
 System.out.println(rno+"\t"+name+"\t"+address);
 }
}
```

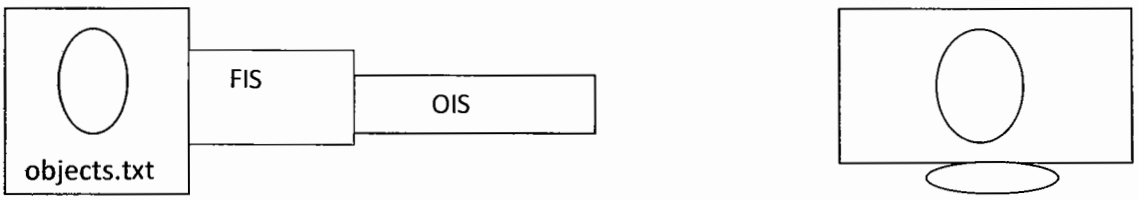
ObjectWriting.java (serialization)

```
import java.io.*;
class ObjectWriting{
public static void main(String args[]) throws IOException{
FileOutputStream fos = new FileOutputStream("objects.txt");
ObjectOutputStream oos = new ObjectOutputStream(fos);
 Student s = new Student(7,"sachin","delhi");
 oos.writeObject(s);
 System.out.println("Object is Written...");
 oos.close();
}}
```



ObjectReading.java (de-serialization)

```
import java.io.*;
class ObjectReading{
public static void main(String args[]) throws IOException,ClassNotFoundException{
FileInputStream fis = new FileInputStream("objects.txt");
ObjectInputStream ois = new ObjectInputStream(fis);
 Student s = (Student) ois.readObject();
 s.display();
/*
//RTE: EOFException
 s = (Student) ois.readObject();
 s.display();
*/
 ois.close();
 }
}
```



// wap to write multiple objects onto object files
// wap to read multiple objects from object files

transient keyword

-we can transfer an object from one location to another location only when corresponding object is serialized.
-when we transfer an object from one location to another location then all the object data will be transferred
-but if we don't want to transfer few values of the object then the variables are declared using a java keyword or modifier called " transient " which are called transient variables
- transient variables are the variables which can not participate in the serialization.
- when we transfer the transient variables then it will hide the original value and transfer the default values

Rules

-there is no effect using transient keyword on static variables because static variables are class variables and they never participate in serialization.
-there is no effect using transient keyword on final variables because final variables are always participate in serialization
-transient keyword can be applied only on instance variables
-transient keyword can not be applied on classes, methods, interfaces, local variables

Console(1.6)

-This class is used to read the data from keyboard or write the data on to monitor.
-This Class will be used in Console based applications(which are running on command prompt)

-This class is introduced in java 1.6 version
-This class contains a special method called readPassword()
-We can not create an object for Console class directly because it contains a private constructor.
-But we can create an object for Console class using a method called System.console()

creation of Console:

```
Console c = System.console();
```

```
//wap to demo on Console class
import java.io.*;
class ConsoleDemo{
 public static void main(String args[]){
  Console c = System.console();
  String user = c.readLine("Enter User Name:");
  char[] pwd = c.readPassword("Enter Password:");
  c.printf("\nUser Name:"+user);
  c.printf("\nPassword:"+pwd);
 }
}
```

Threads

single tasking

single tasking means executing 1 task at a time. Here much of the processor time will be wasted because waiting time very high and processor gives response late.

Eg:
DOS

multi tasking

multi tasking means executing multiple tasks at the same time simultaneously. Here processor time utilized in a proper manner and performance of the application by reducing waiting time and processor gives response faster.

Eg:
windows, student,.....

- we can achieve multitasking in following 2 ways

1. Processor based multi tasking

- Processor based multi tasking means executing multiple tasks at the same time simultaneously where each task is independent of other tasks having separate memory and resources
- Processor based multi tasking is an operating system approach

Eg:
java program,
listen to music,
download songs,
copy softwares,
chating,
....

2. Thread Based multi tasking

-Thread based multi tasking means executing different parts of the same program at the same time simultaneously where each task is independent sometimes or dependent sometimes of other tasks which are having common memory and resources.
-Thread based multi tasking is programming approach
-Each different of the program is called Trheads

Eg:
games, web based apps,....

Time slice/ time quantum

For executing each task a unit of CPU time is given for its execution which is called time slice or time quantum.

Thread Scheduler

Thread Scheduler the operating system program which is responsible for allocating the resources for threads and executing them.

Context Switching

Context Switching is the process of loading or unloading the processes or tasks into or from CPU

Multi Threading

Multi Threading means executing multiple threads at the same time simultaneously. where each thread is called a separated task of the program that executes separately.

-In Java to work the threads we can use following

java.lang.Thread class

java.lang.Runnable interface

Main Thread

-For every java program there will be created a default thread by JVM which is nothing but Main Thread.

-The entry point for Main Thread is main() method.

```
// wap to display the info of currently Running Thread
class Thread1{
public static void main(String args[]){
Thread ct = Thread.currentThread();
System.out.println(ct);
System.out.println(ct.getName());
System.out.println(ct.getPriority());
System.out.println(ct.getThreadGroup());
}
}
```

here Thread.currentThread() method gives the information of the currently running thread like Thread name, Thread priority and ThreadGroup name in which it belongs to.

user defined threads

-In java we can also create user defined Threads by using any one of the following 2 approaches.

1.by extending the Thread class

2.by implementing Runnable interface

creating the user defined Thread by extending the Thread class

if we want to create the user defined Threads using this approach we have to follow following 5 steps

step1: creating a class that extends a Thread class

```
class MyThread extends Thread{
  }
```

step2: overriding the run() method

```
public void run(){
  //statements;
}
```

here run() method is called entry-point for the user defined thread and all the job done by the user.defined thread will be written here itself.

step3: creating an object for user defined Thread class

```
MyThread mt = new MyThread();
```

step4: attaching user defined thread with main ThreadGroup

```
Thread t = new Thread(mt);
```

step5: starting the execution of the Thread by calling start()

```
t.start();
```

when we call start() method the thread will be registered with ThreadScheduler and next will invoke the run() method.

```
// wap to create the user defined thread using approach1
class MyThread extends Thread{
 public void run(){
  for(int i=1;i<=10;i++){
   System.out.println("User Thread Value:"+i);
  }
 }
}
class Thread2{
 public static void main(String args[]){
  MyThread mt = new MyThread();
  Thread t = new Thread(mt);
  t.start();
 }}
```

creating userdefinedThread by implementing Runnable interface

- if we want to create the user defined Threads using this approach we have to follow following 5 steps

step1: creating a class that implements Runnable interface

```
class MyThread implements Runnable{
  }
```

step2: overriding the run() method

```
public void run(){
  //statements;
 }
```

step3: creating an object for user defined Thread class

```
MyThread mt = new MyThread();
```

step4: attaching user defined Thread with main ThreadGroup

```
Thread t = new Thread(mt);
```

step5: starting user defined Thread by calling start() method

```
t.start();
```

Eg:

```
//wap to define user defined Threads using approach2
class MyThread implements Runnable{
 public void run(){
 for(int i=1;i<=10;i++){
  System.out.println("User Thread Value:"+i);
  }
 }}
class Thread3{
public static void main(String args[]){
 MyThread mt = new MyThread();
  Thread t = new Thread(mt);
  t.start();
 }
}
```

?what is the difference between extending Thread class and implementing Runnable interface

if we create any thread by extending Thread class then we have no chance for extending from any other class. but if we create any thread by implementing Runnable interface then we have a chance for. extending from any one class. it is always recommended to create the user defined threads by implementing Runnable interface.

only.

? can we call run() method directly

(or)

? what is the difference between calling t.run() and t.start()

-Yes we can call run() method directly code is valid but no thread will be created/registered with Thread scheduler but run()method executes like a normal method by Main Thread.

- But if we call start() method thread will be registered with threadscheduler and calls run().

Eg:

```
class  MyThread  implements Runnable{
 public  void  run(){
 Thread t = Thread.currentThread();
    for(int i=1;i<=10;i++){
       System.out.println(t.getName()+" value: "+i);
     }
  }
}
class  Thread4{
public  static  void  main(String args[]){
 MyThread  mt = new MyThread();
 Thread t  = new Thread(mt);
// t.run();
   t.start();
 }
}
```

In the above program if we call t.start() then thread will be created and the output will be displayed like follows

```
Thread-0 Value:1
Thread-0 Value:2
Thread-0 Value:3
.............
```

here Thread-0 is default thread name assigned for user defined thread bydefault.

In the above program if we call t.run() then thread willnot be created and the output will be displayed like follows

```
main Value:1
main Value:2
main Value:3
.........
```

```
//wap to demo on multiple Threads which are acting on
  different  objects and having same job
class  MyThread implements Runnable{
public void run(){
Thread t  = Thread.currentThread();
  for(int i=1;i<=10;i++){
   System.out.println(t.getName()+" Thread value:"+i);
    }
}
}
```

```
class MultiThreading1{
public static void main(String args[]){
  MyThread mt1 = new MyThread();
  MyThread mt2 = new MyThread();
  MyThread mt3 = new MyThread();
Thread t1 = new Thread(mt1);
Thread t2 = new Thread(mt2);
Thread t3 = new Thread(mt3);
  t1.start();
  t2.start();
  t3.start();
}
}
//wap to demo on multiple Threads which are acting on
   different objects and having different job
class EvenThread implements Runnable{
public void run() {
  for(int i=0;i<=20;i=i+2){
    System.out.println("Even Thread value: "+i);
   try{
     Thread.sleep(1000);
     }
   catch(InterruptedException ie){
     }
   }
 }
}
class OddThread implements Runnable{
public void run(){
  for(int i=1;i<=20;i=i+2){
    System.out.println("Odd Thread value: "+i);
  try{
   Thread.sleep(1000);
    }
  catch(InterruptedException ie){
    }
   }
 }
}
class MultiThreading2{
public static void main(String args[]) throws InterruptedException{
 EvenThread et = new EvenThread();
 OddThread ot = new OddThread();
 Thread t1 = new Thread(et);
 Thread t2 = new Thread(ot);
  t1.start();
  t2.start();
for(int i=1;i<=10;i++){
   System.out.println("Main Thread value: "+i);
```

```
      Thread.sleep(1000);
   }
  }}
```

Note:

When we execute multiple threads at the same time simultaneously then we never get same output for every execution because Thread Scheduler will decide which thread should execute in next step.

```
//wap to create multiple threads which acting on same object
class CollegeThread implements Runnable{
int seats;
CollegeThread(int seats){
this.seats=seats;
}
public void run(){
Thread ct = Thread.currentThread();
String tname = ct.getName();
System.out.println(tname+"No.of Seats available before allotment:"+seats);
if(seats>0){
try{
Thread.sleep(1000);
}
catch(InterruptedException ie){
}
System.out.println("Seat is Alloted to"+tname);
seats=seats-1;
}
else{
System.out.println("Seat is Not Alloted to"+tname);
}
System.out.println(tname+"No.of Seats available after allotment:"+seats);
}}
class Allotment{
public static void main(String args[]) {
 CollegeThread ct = new CollegeThread(1);
Thread t1 = new Thread(ct);
Thread t2 = new Thread(ct);
Thread t3 = new Thread(ct);
t1.setName("sachin");
t2.setName("sehwagh");
t3.setName("dhoni");
  t1.start();
  t2.start();
  t3.start();
 }
}
```

Note:

-In the above program when we have 50 seats then the seat is allotted for all the 3 people.

-But when we have only 1 seat then only one person has to get the seat, but seat is again allotted for all the 3 people which is nothing but " data inconsistency problem".

data inconsistancy problem

-when we execute multiple Threads which are acting on same object at the same time simultaniously then there is chance of occurring data inconsistency problem in the application

-data inconsistency problem will occur because one thread is updating the value at the same time other thread is using old value.

-To resolve this data inconsistency problem we have to synchronize the object on which multiple Threads are acting.

Thread synchronization

-Thread synchronization is a process of allowing only one thread to use the object when multiple Threads are trying to use the particular object at the same time.

-To achieve this Thread synchronization we have to use a java keyword or modifier called " synchronized ".

-We can achieve Thread synchronization in following 2 ways

1. synchronized blocks

If we want to synchronize a group of statements then we have to go for synchronized blocks

syntax:

```
synchronized(object) {
  //statements
  }
```

2. synchronized methods

If we want to synchronize all the statements of the particular method then we have to go for synchronized methods.

syntax:

```
synchronized returntype methodname(parameters){
  //statements;
 }
```

Note:

-In the last program we are running multiple Threads which are acting on same object at the same time simultaneously so that there is a chance of occurring data inconsistency problem (not a thread safe program). `

-But if we want to resolve the problem of data inconsistency problem and make the program a thread-safe one, then we have to use thread synchronization concept like follows

```
synchronized(this){
 if(seats>0){
  }
 else{
  }
 }
```

or

```
synchronized public void run(){
     //statements
  }
```

Note:

Thread synchronization concept is recommended to use only when we run multiple threads which are acting on same object otherwise this concept is not required.

deadlock

-When we execute multiple Threads which are acting on same object that is synchronized at the same time simultaneously then there is another problem may occur called deadlock

-Dead lock may occur if one thread holding resource1 and waiting for resource2 release by the Thread2, at the same time Thread2 is holding on resource2 and waiting for the resource1 released by the Thread1 in this case 2 Threads are continuously waiting and no thread will execute this situation is called as deadlock.

-To resolve this deadlock situation there is no any concept in java, programmer only responsible for writing the proper logic to resolve the problem of deadlock.

Thread class

Thread is a predefined class available in java.lang package which is used to create the Threads, execute the Threads and manipulate the Threads.

creation of Thread

Thread t = new Thread();
Thread t = new Thread(String tname);
Thread t = new Thread(Runnable obj);
Thread t = new Thread(Runnable obj,String tname);
all these constructors will create the new Thread and attach with main ThreadGroup bydefault.

Thread t=new Thread(ThreadGroup tg);
Thread t=new Thread(ThreadGroup tg,String tname);
Thread t=new Thread(ThreadGroup tg,Runnable obj);
Thread t=new Thread(ThreadGroup tg,Runnable obj,String tname);
all these 4 constructors will create the new Thread and attach with user defined ThreadGroup.

Methods of Thread class

1. Thread currentThread()

this method used to return the information of the currently running Thread like thread name, thread3 priority, thread group name in which it belongs to.

2. String getName()

this method returns the name of the particular thread , we can get the name of any thread like main thread or user defined thread.

3. void setName(String tname)

this method change the name of the particular thread, we can change the name of any thread that it may be main thread or user defined thread.

4. int getPriority()

this method returns the priority of the particular thread , we can get the priority of any thread that is main thread or user defined thread.

5. void setPriority(int priority)

this method is used to change the priority of the particular thread, we can change the priority of any thread that it may be main thread or user defined thread.

priority:

-priority means the amount of resources allocated to the particuler thread.
-the default priority of main Thread is 5-child thread will take the priority that is equal to its parent thread priority
-we can change the priority of any thread that it may be main thread or user defined thread.
-it is recommended to change the priority by using constants available in the Thread class like follows

Thread.MIN_PRIORITY;
Thread.NORM_PRIORITY;
Thread.MAX_PRIORITY;

Eg:
t.setPriority(7) - valid
t.setPriority(Thread.NORM_PRIORITY+2)-valid(recommended)

Note:

1.According to sun micro system's jvm the range of thread priority is 1-10
2.The range of thread priorities will be changed from jvm to jvm
3.If the specified priority value is not in the range of priority then setPriority() method throws a runtime exception called IllegalArgumentException.

6. void sleep(long ms)

This method will make particular thread to wait for specified milli seconds of time. this method will throw a compile time exception called "InterruptedException" which must be handled

7. void start()

This method is used to start the execution of the thread and when we we call start() method it will perform following 2 operations

1.Thread will be registered with Thread scheduler, once the Thread get registered with Thread scheduler, it will

provide resources required by the Thread
2. it will invoke the run() method

Note:

For a Thread we can call start() method at most 1 time because thread dont need to register with Thread Scheduler many times otherwise if we call start() method for many times for a particular Thread it will throw a Runtime exception called " IllegalThreadStateException "

8. ThreadGroup getThreadGroup()

-this method returns name of the ThreadGroup in which thread belongs to.
-bydefault all the Threads are attached with main ThreadGroup
-we can also create our own Thread Group by taking the help of ThreadGroup class

creation of ThreadGroup:

ThreadGroup tg = new ThreadGroup("name");

-Main advantage of ThreadGroup is we can attach multiple Threads so that we can communicate with multiple Threads at a time

Eg:

```
class MyThread implements Runnable{
 public void run(){
 Thread ct = Thread.currentThread();
  System.out.println(ct);
   }
}
class ThreadDemo{
 public static void main(String args[]){
  MyThread mt = new MyThread();
ThreadGroup tg = new ThreadGroup("UDTG");
 Thread t1 = new Thread(mt);
 Thread t2 = new Thread(tg,mt);
 Thread t3 = new Thread(mt);
 Thread t4 = new Thread(tg,mt);
 t1.start();
 t2.start();
 t3.start();
 t4.start();
 Thread ct = Thread.currentThread();
 System.out.println(ct);
 }
}
```

9. boolean isAlive()

- this method returns true if the Thread is alive otherwise it returns false.
- the thread said to be alive until it executes the run().
- once it executes the last statement in the the run() method then thread goes to dead state.

10. void join()

This method make the parent thread to wait until the completion of child threads. This method also throw a compile time exception called InterruptedException which must be handled

```
//wap to demo on join() method
class MyThread implements Runnable{
int n;
```

```
MyThread(int n){
this.n=n;
}
public void run(){
Thread t  = Thread.currentThread();
System.out.println(t.getName()+" Thread is started....");
 for(int i=1;i<=n;i++){
   System.out.println(t.getName()+" Thread value:"+i);
    try{
Thread.sleep(1000);
}
catch(InterruptedException  ie){
}
   }
System.out.println(t.getName()+" Thread is Terminated....");
}
}
class  JoinDemo{
public static  void  main(String  args[]){
   System.out.println("Main Thread is started....");
  MyThread mt1= new  MyThread(10);
  MyThread mt2= new  MyThread(15);
 Thread t1 = new Thread(mt1,"child1");
 Thread t2 = new Thread(mt2,"child2");
  t1.start();
  t2.start();
  for(int i=1;i<=5;i++){
   System.out.println("Main Thread value:"+i);
    try{
        Thread.sleep(1000);
     }
   catch(InterruptedException  ie){
     }
   }
 System.out.println(t1.isAlive());//true
 System.out.println(t2.isAlive());//true
try{
t1.join();
t2.join();
}
catch(InterruptedException  ie){
}
 System.out.println(t1.isAlive());//false
  System.out.println(t2.isAlive());//false
  System.out.println("Main Thread is Terminated....");
 }
}
```

11. boolean isDaemon()

this method returns true if the particular Thread is Daemon Thread otherwise it returns false.

12. void setDaemon(boolean)

this method is used to change the threads as daemon threads or non-daemon threads.

Daemon Threads

- Daemon Threads are called background Threads which work in the background and provide the service for other threads.

-Daemon Threads will run only when other threads are available otherwise JVM will shut down all the daemon threads.

-Bydefault main thread and all its child threads are non-daemon Threads. but if we want to make the ser defined thread as daemon thread we have to use setDaemon() method

Eg:

garbage collector

```
// wap to demo on  Daemon Threads
class MyThread implements Runnable{
double num;
 public void run(){
  for(int i=1;i<=10000;i++){
  num = Math.random();
   //System.out.println("User Thread Value:"+num);
  try{
   Thread.sleep(1000);
    }
  catch(InterruptedException ie){
   }
  }
 }
}
class DaemonThreadDemo{
public static void main(String args[]){
MyThread mt = new MyThread();
 Thread  t = new Thread(mt);
 t.setDaemon(true);//making the user thread as daemonthread
  t.start();
for(int i=1;i<=5;i++){
  System.out.println("Main Thread Value:"+mt.num);
 try{
  Thread.sleep(1000);
   }
 catch(InterruptedException ie){
  }
 }
}
}
```

Note:

1.we must call t.setDaemon(true) method before starting the thread otherwise it will throw a run time exception called IllegalThreadStateException and user thread wont become as Daemon Thread

2. we can not change our main Thread as daemon thread.

Methods of Object class which are related to threads

1. wait():

This method used to make the particular Thread wait until it gets a notification.

2. notify():

This method used to send the notification to one of the waiting thread so that thread enter into running state and execute the remaining task.

3. notifyAll()

This method used to send the notification to all the waiting threads so that all thread enter into running state and execute simultaneously.

- all these 3 methods are available in Object class which is super most class so that we can access all these 3 methods in any class directly with out any reference.
- these methods are mainly used to perform Inner thread communication
- by using these methods we can resolve the problems like Producer-Consumer problem, Reader -Writer problem,...

```
// Wap to demo on InnerThread Communication
class MyThread implements Runnable{
 int total;
 public void run(){
  for(int i=1;i<=1000;i++){
    total=total+i;
   if(total>10000){
   synchronized(this){
    notify();
     }
   }
   try{
    Thread.sleep(5);
    }
   catch(InterruptedException ie){
    }
 }
System.out.println("User Thread total:"+total);
 }
}
class InnerThreadComm{
public static void main(String args[]){
 MyThread mt = new MyThread();
 Thread t = new Thread(mt);
 t.start();
try{
   synchronized(mt){
    mt.wait();
     }
  }
catch(InterruptedException ie){
 }
 System.out.println("Main Thread total:"+mt.total);
}
}
```

Note:

we must call wait(), notify(), notifyAll() methods in side the synchronized blocks or synchronized methods otherwise it will throw a runtime exception called IllegalMonitorStateException

```
//wap to solve Producer-Consumer problem
class ShowRoom{
```

```
 int value;
 boolean pro_thread=true;
synchronized void produce(int i){
  if(pro_thread==true){
    value=i;
System.out.println("Procuder Has Produced Product "+value);
    pro_thread=false;
   notify();
    }
 try{
     wait();
       }
 catch(InterruptedException ie){
 }
 }
synchronized int  consume(){
   if(pro_thread==true){
 try{
    wait();
  }
 catch(InterruptedException ie){
 }
    }
    pro_thread=true;
     notify();
     return value;
}
}
class Producer  implements Runnable{
  ShowRoom  sr;
Producer(ShowRoom  sr){
 this.sr=sr;
}
public void run(){
 int  i=1;
while(true){
 sr.produce(i);
 try{
    Thread.sleep(1000);
    }
   catch(InterruptedException ie){
    }
 i++;
}
}
}
class  Consumer  implements Runnable{
  ShowRoom  sr;
Consumer(ShowRoom  sr){
```

```
 this.sr=sr;
}
public void run(){
while(true){
 int res =  sr.consume();
System.out.println("Consumer Has Taken Product "+res);
  try{
     Thread.sleep(1000);
     }
   catch(InterruptedException ie){
     }
}
}
}
class ProducerConsumer{
public  static  void  main(String args[]){
   ShowRoom  sr = new ShowRoom();
    Producer  p =  new Producer(sr);
    Consumer c =  new Consumer(sr);
   Thread  t1 = new Thread(p);
   Thread  t2 = new Thread(c);
    t1.start();
    t2.start();
  }
}
```

Types of Threads

1.orhpan thread

if any thread running without any help of its Paremt Thread.then it is called as orphan thread.
to create this kind opf threads we can use join() method.

2. helper threads

if any thread waiting and giving a chance to other threads to execute then these threads are called helper threads when we call sleep() or join() or wait() method then that waiting thread is called as helper thread.

3. selfish Threads

if any thread is running until the completion of its job or executing completely then it is called as selfish thread.

4. starving threads

if any thread waiting for a long time then it is called as starving threads.
generally we can observe this kind of selfish threads or starving threads in synchronization or when we use wait() and notify() methods

5. Green Threads

Green Threads are called JVM level Threads which are used allocate the resources to other threads.

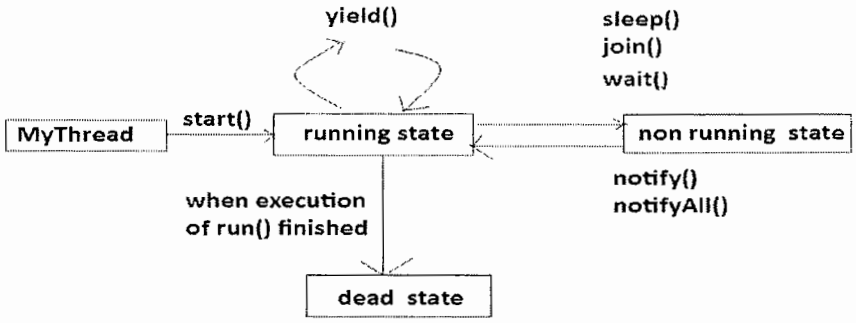
6. Native Threads

Native Threads are called as OS level Threads which are also responsible for allocating the resources to other threads. Here allocation of resources is efficient than Green Threads.

7. Daemon Threads

Daemon Threads are the background threads which work in the background and provide service for other threads

Thread life cycle



Nested classes or Inner classes

-If a class is declared with in another class then this concept is called as Inner classes or nested classes

-If a class contains other class then it is called as outer class or top level class

- class declared inside the outer class is called as inner class

syntax:

```
class A{
  class B{
    }
  }
```

Here

A - outer class or top level class B - inner class

-Here outer class can be default or public only but not private or protected or static

-But inner class can be default or public or private or protected or static

-Te main advantage of Inner classes is that we can access the members of (both instance and static) Outer class inside the inner class directly without taking the help of any object.

-inner classes can also be called as helper classes

-inner classes are hidden from other class of same package or other package

we have following 2 types of inner classes

1. non-static Nested classes or Inner classes
2. static Nested classes

1. non-static nested class or inner classes

- If any inner class is created without using any static keyword then it is called as non-static Nested classes or Inner classes

- Non-static Nested or Inner classes can contain only non-static members(instance members)

- based on the place where we declare these inner classes are again classified into following 3 types

1. Regular Inner class (member level)
2. Method local Inner class (statement level)
3. Anonymous Inner class(expression level)

1. Regular Inner class

-If an inner class is declared outside the methods of Outer class then it is called as Regular Inner class.

-Regular Inner class are always created at member level

-Regular Inner class can be default or public or private or protected

syntax:

```
class Outer{
  class Inner{
    }
```

```
 void outerMethod(){
  }
}
```

if we compile the above program compiler will generate .class for both outer class and inner class available in the program like follows,

1 Outer.class 2. Outer$Inner.class

Accessing the members of Regular Inner class

-if we want to access the members of Regular Inner class we must create an object for Regular Inner class.

-we can create an object for Regular Inner class in following different ways,

creating an object for Regular Inner class inside the instance methods of Outer class

```
class Outer{
 class Inner{
  void innerMethod(){
   System.out.println("Inner class innerMethod()");
  }
  }
  void outerMethod(){
    Inner i = new Inner();
    i.innerMethod();
    System.out.println("Outer class outerMethod()");
  }
 public static void main(String args[]){
   Outer o = new Outer();
   o.outerMethod();
 }
}
```

creating an object for Regular Inner class outside the methods of Outer class

```
class Outer{
  String str="inetsolv";
  Inner i = new Inner();
 class Inner{
  void innerMethod(){
   System.out.println("Inner class innerMethod()");
  }
  }
  void outerMethod(){
   System.out.println("Outer class outerMethod()");
  }
 public static void main(String args[]){
  Outer o = new Outer();
   o.outerMethod();
  System.out.println(o.str);
  System.out.println(o.str.toUpperCase());
  o.i.innerMethod();
 }
}
```

creating an object for Regular Inner class inside the another class by using enclosing class or outer class object.

```
class Outer{
```

```
  class Inner{
    void innerMethod(){
     System.out.println("Inner class innerMethod()");
     }
   }
 void outerMethod(){
   System.out.println("Outer class outerMethod()");
  }
 /*
public static void main(String args[]){
 Outer o = new Outer();
 o.outerMethod();
Inner i = o. new Inner();
 i.innerMethod();
}
*/
}
class Test{
public static void main(String args[]){
 Outer o = new Outer();
  o.outerMethod();
Outer.Inner i = o.new Inner();
  i.innerMethod();
 }
}
```

accessing the members of Outer class inside inner class

```
class Outer{
 int x=10;
 static int y=10;
 String str="Outer";
class Inner{
 String str="Inner";
  void innerMethod(){
   System.out.println("Inner class innerMethod()");
   System.out.println("x="+x);
   System.out.println("y="+y);
   System.out.println("Addition="+(x+y));
   outerMethod();
  }
  void innerMethod1(String str){
   System.out.println("str="+str);
   System.out.println("str="+this.str);
   System.out.println("str="+Outer.this.str);
  }
 }
 void outerMethod(){
   System.out.println("Outer class outerMethod()");
  }
 }
```

```
class Test{
 public static void main(String args[]){
  Outer o = new Outer();
  Outer.Inner i = o.new Inner();
  i.innerMethod();
  i.innerMethod1("local");
 }
}
```

Note:

-We can directly access the members of outer class (both static and non-static) inside the inner class without taking the help of outer class object.

-To differentiate between the members of Inner class and outer class we have to use this keyword like follows

-- this.member inside the inner class indicates members of Inner class

-- outerclassname.this.member inside the inner class indicates members of Outer class

2. Method local Inner class

-If we declare an inner class inside the methods of Outer class then it is called as Method Local Inner class.

-Method Local Inner classes are always created in side the method at statements level.

-A method local Inner class can not be created as private, public, protected or static

syntax:

```
class Outer{
 void outerMethod(){
    class Inner{
     }
  }
}
```

if we compile the above program then compiler will create following 2 .class files

1. Outer.class 2.Outer$1Inner.class

accessing members of Method Local Inner class

-To access the members of Method local inner class we have to create an object for Method loca inner class

- we can create an object for Method local inner class only in one location that is with in the same method where the inner class declared that too after class declaration

- but we can not create an object in following locations

1. inside other instance methods
2. inside other static methods
3. outside the methods of outer class
4. inside other class
5. with in same method before the class declaration

```
//wap to demo on Method local inner class
 class Outer{
    int x=10;
  static int y=10;
  void outerMethod(){
   System.out.println("Outer class outerMethod()");
    class Inner{
      void innerMethod(){
        System.out.println("Inner class innerMethod()");
        System.out.println("x="+x);
        System.out.println("y="+y);
        System.out.println("Addition="+(x+y));
```

```
        }
      }
     Inner i = new Inner();
      i.innerMethod();
     }
 public static void main(String args[]){
   Outer o = new Outer();
     o.outerMethod();
 }
}
```

3. Anonymous Inner class

-If any inner class is declared with out any name then it is called as Anonymous Inner class

-Anonymous inner classes are always created at expression level

-If we want to create Anonymous Inner classes we have to take the help of any existing class or interface

-At the time of creating the object of Anonymous Inner classes only Anonymous Inner classes will be created.

-For anonymous classes we can create only 1 object because it does't contain any name so that memory can be utilized properly.

syntax:

```
  class Outer{
      new inerfacename()/classname(){
         //members of anonymous Inner class
        };
     }
```

if we compile the above program compiler will generate following 2 .class files

1. Outer.class 2. Outer$1.class (anonymous class)

Creating Anonymous Inner class using interface

```
class Outer{
public static void main(String args[]){
 Runnable r = new Runnable(){  //anonymous Inner class
  public void run(){
  Thread t = Thread.currentThread();
   for(int i=1;i<=10;i++){
    System.out.println(t.getName()+" value:"+i);
   }
  }
 };
Thread t = new Thread(r);
  t.start();
 }
}
```

creating Anonymous Inner class using class

```
class Outer{
public static void main(String args[]){
  Thread mt = new Thread(){     //anonymous Inner class
    public void run(){
     Thread t = Thread.currentThread();
      for(int i=1;i<=10;i++){
      System.out.println(t.getName()+" value:"+i);
       }
```

```
        }
    };
    Thread t = new Thread(mt);
    t.start();
   }
}
```

creating Anonymous Inner class while passing the value

```
class Outer{
 public static void main(String args[]){
//anonymous Innerclass
Thread t = new Thread(new Runnable(){
     public void run(){
      Thread t = Thread.currentThread();
       for(int i=1;i<=10;i++){
       System.out.println(t.getName()+" value:"+i);
       }
    }
  });
  t.start();
 }
}
```

2. static Nested classes

- If we declare any nested class using static key word then it is called as static Nested classes
-We can create static regular classes but we cannot create any method local static or anonymous static nested classes
-static nested classes can have both static and non-static members

syntax:

```
class Outer{
 static classs Inner{
     }
  void outerMethod(){
   }
 }
```

1. Outer.class 2. Outer$Inner.class

```
//wap to demo on static nested classes
class Outer{
int x=10;
static int y=20;
 static class Inner{
    void innerMethod(){
      System.out.println("Inner class innerMethod()");
      //System.out.println("x="+x); -invalid
         System.out.println("y="+y);
       }
  }
 void outerMethod(){
    System.out.println("Outer class outerMethod()");
   }
 }
class Test{
```

```
 public static void main(String args[]){
    Outer o = new Outer();
     o.outerMethod();
  Outer.Inner i = new Outer.Inner();
     i.innerMethod();
  }
}
```

IQ: can we write main() in Inner classes

Yes we can write a main() method inside the static nested class because it contains both static and non-static members, but we can not write main() method in side the non-static nested class because non-static nested classes can contain only non-static memebers but main() method is static.

Eg:

```
class Outer{
 static class Inner{
   public static void main(String args[]){
    System.out.println("Inner class main() Method");
   }
 }
 public static void main(String args[]){
   System.out.println("Outer class main() Method");
 }
}
```

Exceution

```
  java Outer
o/p:
  Outer class main() Method()
  java Outer$Inner
o/p:
 Inner class main() Method()
```

AWT

*__AWT__: (Abstract Windowing Toolkit)

When a user want to interact with an application, the user has to provide some information to the application and it can be done in two ways.

__1.Character user Interface(CUI)__:

-This Interface is use to interact with the application by typing some characters.

-This interface is not user friendly because the user has to type all the commands and the user has to remember all the commands.

Eg: DOS (character user Interface)

__2.Graphical user Interface(GUI)__:

-This Interface will interact with the applicationby using some graphics like menu's, icons ,images,...

-This Interface is user friendly because it prompts the user by providing the options (or) menus.

__Eg:__ window xp, windows 7(os)

- To develop the GUI based applications we have to use AWT.

-AWT stadns for abstract windowing toolkit.

-The set of classes & Interfaces which are required to develop. GUI components together are called as 'Toolkit'. The GUI components will be used to design GUI programs.

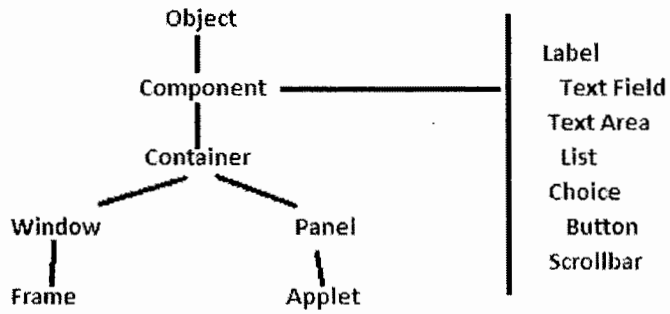
-Writing a program to display the created GUI components on the windows is called as 'windowing'.

-To display the components on the windows we need to take the support of graphics available in the operating system for a developer there is no direct interaction with the graphics and hence graphics is 'Abstract' to the developer.

-Every GUI component will have a corresponding 'PEER' class which is responsible to interact with the graphics of the 'operating system'.

here Collection of the all peer class is called as a 'PEER SERVICE'.

__AWT HIERATCHY__:



__Frame__:

-A frame is a Rectangular Box containing the borders.

-Every frame will by default contain a title, minimize, maximize and close Buttons

-By default Frame is invisible (false)

-By default the size of frame OXO (zero by zero) pixels.

-The frame can be created in two ways.

1. Create the object of Frame class directly.

Eg: Frame f = new Frame();

2. Create the object of any class that extends Frame class.

Eg: class MyFrame extends Frame

MyFrame mf = new MyFrame();

```
//Program to create a frame using the first technique
import java.awt.*;
public static void main(String[] args){
//Creation of frame object directly
Frame f = new Frame();
//making the frame visible
f.setVisible(true);
//setting the size of the frame
f.setSize(300,300);
   }
}
//program to create a frame using the second technique
import java.awt.*;
class FrameDemo extends Frame{
public static void main(String[] args){
//creation of frame object directly
FrameDemo fd = new FrameDemo();
//making the frame visible
fd.setVisible(true);
//setting the size of the frame
fd.setSize(500,200);
        }
  }
```

AWT Events:

-An event is used for making communication or interacting with the 'GUI' components.

-Events will be generated or triggered automatically based on the user operation.

Eg: Left click, Right click and Double click, typing some text, selecting some values etc

-An event is an object which contains the information of the event that is generated and also contains the Information of the component that has generated the Event.

-To perform any operation the component has to listen to the event that is generated.

-But the component can not listen to the events that are generated.

-to sole this problem we take the help of the listeners which are interfaces which can listen to the events that are generated.

-After the listener listens to the generated event, it delegates (passes) the event information to one of the methods available in that listener. This procedure is called as "Event Delegation Model".

-They are different types of listeners areavaialble which can listen to their corresponding events.

procedure to use a listener in an application:

1.Choose a listener (interface) appropriate to the application requirement.

2.Once listener is selected we have to provide implementation for all the methods of particualr listener

3. register the listener with the particular component using following mehod

compobj.addXXXListener(XXXListener obj)

```
import java.awt.*;
import java.awt.event.*;
class FrameDemo extends Frame Implements WindowListener{
FrameDemo(){
//making the frame visible
setVisible(true);
//setting the size of the frame
setSize(300,300);
//setting the title of the frame
setTitle("Window Listener");
//adding the listener to the frame
addWindowListener(this);   }
public static void main(String[] args){
//creation of frame object
FrameDemo fd = new FrameDemo();
    }
Public void windowOpened(WindowEvent we){   }
Public void windowClosing(WindowEvent we){
System.exit(0);
      }
Public void windowClosed(WindowEvent we){     }
Public void windowActivated(WindowEvent we){     }
Public void windowDeativated(WindowEvent we){     }
Public void windowIconified(WindowEvent we){     }
Public void windowDeiconified(WindowEvent we){     }
  }//end of the class
```

XXXAdapter classes:

-This calss is an implementation class of XXXListener which provides implementioan(null) for allthe mehods of particular XXXListener

-Adaper classes can be used to override the required methods of particular XXXListener

Eg:

MouseAdapter,keyAdapter,WindowAdapter

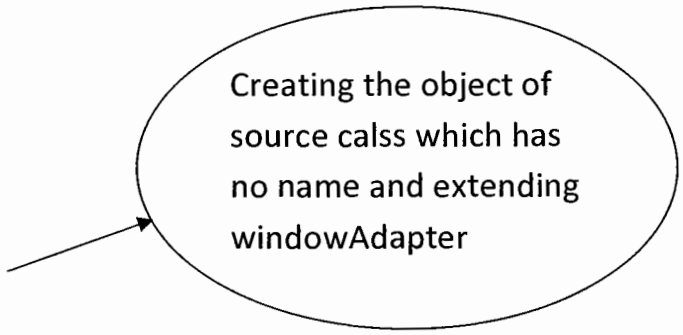
```
//wap to cliose the windiow using WindowAdapter
import java.awt.*;
import java.awt.event.*;
class FrameDemo extends Frame{
FrameDemo(){
this.setVisible(true);
this.setSize(300,500);
this.setTitle("windowAdapter");
addWindowListener(new MyFrame());
      }
Public static void main(String[] args){
```

```
FrameDemo fd = new FrameDemo();
      }
}
class MyFrame extends WindowAdapter{
public void windowClosing(WindowEvent we){
System.exit(0);
        }
 }
```

Inner class: If a class declared inside another class then it is called Inner class.

Anonymous inner class: An inner class which does not contains any name is called as Anonymous inner class.

```
import java.awt.*;
import java.awt.event.*;
class FrameDemo extends Frame{
FrameDemo(){
this.setVisible(true);
this.setSize(300,300);
this.setTitle("Anonymous inner class");
addWindowListener(new WindowAdapter(){
public void windowClosing(WindowEvent ae){
System.exit(0);
            }
       });
  }
Public static void main(String[] args){
FrameDemo fd = new FrameDemo();
       }
}
```



drawString() Method: This method used to display a message on the frame.

Syntax: drawString(String,x,y);

The string that is specified will be displayed (x,y) location of the frame.

The drawstring method belong to graphics class and we can get the reference of graphics class by using paintmethod.

Syntax: public void paint(Graphics g)

The paint() will be invoke automatically when the frame is created and loaded.

```
//program will be display on
import java.awt.*;
import java.awt.event.*;
class TextFrame extends Frame{
TextFrame(){
SetVisible(true);
SetSize(300,300);
SetTitle("message frame");
addWindowListener(new WindowAdapter(){
public void windowClosing(WindowEvent we){
System.exit(0);
```

```
            }
        });
}
public static void main(String[] args){
new TextFrame();
            }
public void paint(Graphics g){
//creation a color
Color c = new Color(200,0,0);
//setting the color
g.setColor(c);
//creating a font
Font f = new Font("arial", Font.BOLD,34);
//setting the font
g.setFont(f);
//displaying a message on the frame
g.drawString("Hello students",100,100);
            }
}
```

Button: This component can be use to perform some operation when the user clicks on a button.

Creation of button: Button b = new Button(String label);

```
import java.awt.*;
import java.awt.event.*;
class ButtonDemo extends Frame{
//declaring the components
Button b1,b2;
ButtonDemo(){
//creating the components
b1 = new Button("OK");
b2 = new Button("CANCEL");
//setting the layout to null layout
this.setLayout(null);
//setting the boundaries for the components
b1.setBounds(100,100,80,40); //(x, y, w, h)
b2.setBounds(200,100,80,40); //(x, y, w, h)
//adding the component to the frame
this.add(b1);
this.add(b2);
this.setVisible(true);
this.setSize(300,300);
this.setTitle("button");
this.addWindowListener(new WindowAdapter(){
public void windowClosing(WindowEvent we){
System.exit(0);
                }
```

```
        });
 }
Public static void main(String[] args){
New ButtonDemo();
        }
}
```

Label: This component is used to display a message on the frame. This component will be generally used along with other components.

Creation of Label: Label l = new Label(String);

TextField: This component will allow the user to enter some text.

Creation of TextField: TextField tf = new TextField(size);

```
import java.awt.*;
import java.awt.event.*;
class LoginDemo extends Frame{
Label userl, pwdl;
TextField usertf, pwdtf;
LoginDemo(){
userl = new Label("user_Name");
Pwdl = new Label("password");
Usertf = new TextField(20);
Pwdtf = new TextField(20);
SetLayout(null);
userl.setBounds(100,100,80,40);
usertf.setBounds(200,100,80,30);
pwdl.setBOunds(100,200,80,30);
pwdtf.setBounds(200,200,80,30);
this.add(userl);
this.add(usertf);
this.add(pwdl);
this.add(pwdl);
this.setVisible(true);
this.setSize(400,500);
this.setTitle("TextField Label");
this.addWindowListener(new WindowAdapter(){
public void windowClosing(WindowEvent we){
System.exit(0);
          }
    });
 }
public static void main(String[] args){
new LoginDemo();
       }
}
```

Checkbox: This component allows the user select any number of options from an even group of options.

Creation of checkbox:

Checkbox cb = new Checkbox(Label);

RadioButton: This component will allow the user to select any one option from group of options. To place the options under single group we are support to use class called "CheckboxGroup".

Creation of RadioButton:

```
CheckboxGroup cbg = new CheckboxGroup();
Checkbox rb = new Checkbox(Label, cbg, boolean);
```

Note: To create the RadioButton we use the same Checkbox class.

TextArea: This component will allow the user to write the text in multiple lines.

Creation of TextArea:

```
TextArea ta = new TextArea(rows,cols);
import java.awt.*;
import java.awt.event.*;
class SelectionFrame extends Frame{
Checkbox cb1, cb2, cb3, rb1, rb2, rb3;
CheckboxGroup cbg;
TextArea ta;
SelectionFrame(){
cb1 = new Checkbox("programming");
cb2 = new Checkbox("Reading");
cb3 = new Checkbox("Browsing");
cbg = new CheckboxGroup();
rb1 = new Checkbox("Btech",cbg,false);
rb2 = new Checkbox("BE",cbg,false);
rb3 = new Checkbox("MCA",cbg,false);
ta = new TextArea(6,20);
SetLayout(new FlowLayout());
add(cb1);
add(cb2);
add(cb3);
add(rb1);
add(rb2);
add(rb3);
add(ta);
SetVisible(true);
SetSize(400,500);
SetTitle("SelectionFrame");
addWindowListener(new WindowAdapter(){
public void windowClosing(WindowEvent we){
System.exit(0);
            }
    });
}

public static void main(String[] args){
new SelectionFrame();
```

}}

Choice: This component will display group of times as a drop down menu from which a user can select only one item.

Creation of choice:

Choice ch = new Choice();

To add the items to the choice we use add()

ch.add(item);

List: This component will display a group of items as a scrolling menu from which the user can select any no.of items.

Creation of List:

List l = new List(int, boolean);

We can add the items to the List by using add()

l.add(item);

```
import java.awt.*;
import java.awt.event.*;
class SelectionFrame extends Frame{
Choice ch;
List l;
SelectionFrame(){
ch = new Choice();
ch.add("Hyderabad");
ch.add("pune");
ch.add("Chennai");
ch.add("Noida");
ch.add("Mysore");
ch.add("Bangalore");
l.new List(5,true);
l.add("Hyderabad");
l.add("Pune");
l.add("Chennai");
l.add("Noida");
l.add("Mysore");
l.add("Bangalore");
SetLayout(new FlowLayout());
add(ch);
add(l);
SetVisible(true);
SetSize(500,500);
SetTitle("ChoiceList");
addWindowListener(new WindowAdapter(){
public void windowClosing(WindowEvent we){
System.exit(0);
            }
        });
}
public static void main(String[] args){
new SelectionFrame();
```

```
    }
}
```

*__Listeners__: Listeners are the interfaces which can listen to the Events that are generated.

Listener	Event	Methods
1.ActionListener	ActionEvent	public void actionPerformed(ActionEvent)
2.ItemListener	ItemEvent	public void itemStateChanged(ItemEvent)
3.FocusListener	FocusEvent	public void focusGained/Lost(FocusEvent)
4.MouseMotionListener	MouseEvent	p.v mouseDragged/Moved(MouseEvent)
5.AdjustmentListener	AdjustmentEvent	p.v adjustmentValueChange(AdjEve)

To register the Listeners with the Components we have to use addxxxListener()

__Eg:__ addWindowListener(), addActionListener();,.....

To unregister a Listener with a Components we use removexxxListener()

__Eg:__ removeWindowListener(), removeActionListener(),....

```
//program to use Action Listener an a textFields public.
Public class ListenerDemo Extends Frame implements ActionListener{
Label Userl, Pwdl;
TextField usertf, pwdtf;
ListenerDemo(){
Userl = new Label("username");
Pwdl = new Label("password");
Usertf = new TextField(20);
Pwdtf = new TextField(30);
SetLayout(null);
Userl.setBounds(100,100,80,30);
Usertf.setBounds(200,100,80,30);
Pwdl.setBounds(100,200,80,30);
Pwdltf.setBounds(200,200,80,30);
add(Pwdtf);
add(Userl);
add(pwdl);
add(Usertf);
Usertf.addActionListener(this);
Pwdtf.addActionListener(this);
SetVisible(true);
SetSize(500,500);
SetTitle("MyFrame");
addWindowListener(new WindowAdapter(){
public void windowClosing(WindowEvent we){
System.exit(0);
            }
      });
}
Public static void main(String[] ar){
new ListenerDemo();
```

```
    }
Public void actionPerformed(ActionEvent ae){
String name = Usertf.getText();
String pass = Pwdtf.getText();
Graphics g = this.getGraphics();
            }
   }
//program to demo on Login Applictaion
```

***ActionListener on Button Component**:

```
import java.awt.*;
import java.awt.event.*;
class LoginApp extends Frame implements ActionListener{
Label ul, pl;
TextField utf, ptf;
Button logb;
LoginApp(){
ul = new Label("username");
pl = new Label("password");
utf = new TextField(30);
ptf = new TextField(30);
logb = new Button("Login");
ptf.setEchoChar('*');
this.setLayout(null);
ul.setBounds(100,100,90,30);
utf.setBounds(200,100,90,30);
pl.setBounds(100,150,90,30);
ptf.setBounds(200,150,90,30);
logb.setBounds(150,200,90,30);
this.add(ul);
this.add(utf);
this.add(pl);
this.add(ptf);
this.add(logb);
logb.addActionListener(this);
this.setVisible(true);
this.setSize(300,300);
this.setSize("Listener");
this.addWindowListener(new WindowAdapter(){
public void windowClosing(WindowEvent we){
System.exit(0);
            }
      });
}
Public void actionPerformed(ActionEvent ae){
if(ae.getSource().equals(logb)){ //cancel Button = canb
```

```
String name = utf.getText();
String pwd = ptf.getText();
if(name.equals("i netsolv") & pwd.equals("students"){
new MessageFrame("valid user");
    }
else{
new MessageFrame("Invalid user");
            }
        }
 }
Public static void main(String[] args){
new LoginApp();
    }
}
class MessageFrame extends Frame{
String str;
MessageFrame(String str){
this.str = str;
SetVisible(true);
SetSize(200,200);
SetTitle("Message");
SetBackground(color.yellow);
SetForeground(color.red);
        }
Public void paint(Graphics g){
g.drawString(str,100,100);
        }
  }
//program to implement ItemListener on checkbox & RadioButton
import java.awt.*;
import java.awt.event.*;
class SelectionFrame extends Frame implements ItemListener{
Checkbox jcb,scb,ocb,mrb,frb;
CheckboxGroup cbg;
SelectionFrame;
jcb = new Checkbox("Java");
scb = new Checkbox("Scjp");
ocb = new Checkbox("Oracle");
cbg = new CheckboxGroup();
mrb = new Checkbox("Male",cbg,true);
frb = new Checkbox("Female",cbg,false);
this.setLayout(new FlowLayout());
this.add(jcb);
this.add(scb);
this.add(ocb);
```

```
this.add(mrb);
this.add(frb);
jcb.addItemListener(this);
scb.addItemListener(this);
ocb.addItemListener(this);
mrb.addItemListener(this);
frb.addItemListener(this);
this.setVisible(true);
this.setSize(400,400);
this.setTitle("SelectionFrame");
this.addWindowListener(new WindowAdapter(){
public void windowClosing(WindowEvent we){
System.exit(0);
                }
        });
  }
public void item StateChanged(ItemEvent ie){
repaint ();
    }
Public void paint(Graphics g){
String str = "Course Selected:";
if(jcb.getState()){
str += "scjp";
  }
if(ocb.getState()){
str += "Oracle";
      }
g.drawString(str,100,200);
String msg = "Gender:";
msg += cbg.getSelectedCheckbox().getLabel();
g.drawString(msg,100,250);
        }
Public static void main(String[] args){
new SelectionFrame();
        }
 }
//program to implement ItemListener on choice
import java.awt.*;
import java.awt.event.*;
class SelectionFrame extends Frame implements ItemListener{
Choice ch;
TextArea ta;
Label l;
SelectionFrame(){
ch = new Choice();
```

```
ch.add("BE");
ch.add("BTech");
ch.add("MCA");
ch.add("MBA");
ta = new TextArea(5,30);
l = new Label("Qualification");
SetLayout(new FlowLayout());
add(l);
add(ch);
add(ta);
ch.addItemListener(this);
this.setVisible(true);
this.setSize(500,500);
this.setTitle("SelectionFrame");
this.addWindowListener(new WindowAdapter(){
public void windowClosing(WindowEvent we){
System.exit(0);
                }
        });
   }
public void itemStateChanged(ItemEvent ie){
String option = ch.getSelectedItem();
ta.SetText("Qualification Selected:" + option);
      }
public static void main(String[] args){
new SelectionFrame();
        }
 }
```

***Listeners and Listener Methods**:

Componet	**Listener**	**Listener methods**
1.Button	ActionListener	public void actionPerformed(ActionEvent e)
2.Checkbox	ItemListener	public void itemStateChanged(ItemEvent e)
3.CheckboxGroup	ItemListener	public void itemStateChanged(ItemEvent e)
4.TextField	ActionListener	public void actionPerformed(ActionEvent ae)
	FocusListener	public void focusGained(FocusEvent fe) public void focusLost(FocusEvent fe)
5.TextArea	ActionListener	public void actionPerformed(ActionEvent ae)
	FocusListener	public void focusGained(FocusEvent fe) Public void focusLost(FocusEvent fe)
6.Choice	ActionListener	public void actionPerformed(ActionEvent ae)
	ItemListener	public void itemStateChanged(ItemEvent ie)
7.List	ActionListener	public void actionPerformed(ActionEvent ae)
	ItemListener	public void itemStateChange(ItemEvent ie)
8.Scrollbar	AdjustmentListen	p.v adjustmentValueChange(AdjEvent ae)

	er	
	MouseMotionListener	p.v mouseDragged(MouseEvent me) Public void mouseMoved(MouseEvent me)
9.Frame	WindowListener	public void windowActivated(WindowEvent we) Public void windowClosed(WindowEvent we) Public void windowClosing(WindowEvent we) Public void windowDeactivated(WindowEvent we) Public void windowDeiconified(WindowEvent we) Public void windowIconified(WindowEvent we) Public void windowOpened(WindowEvent we)
10.Keyboard	KeyListener	p.v keyPressed/Released/Type(KeyEvent ke)
11.Label	Nolistener is needed	

JFC-SWING

***Swing**: Swing is use to develop a better efficient GUI.

-The swing components are part of JFC (java foundation classes) which are develop in java.

-The swing components are called as light weight components which will improve the performance of the application because the amount of resources required is very minimum.

-They are no peer classes for the swing components to interact with the operating system.

- Swing component supports pluggable look and feel using which the component can be presented in various flat forms having same behavior.

Note: swing is not a replacement to AWT. But it is an extension.

WindowPane:

-The empty area that is available in a container is which is use for displaying the components is called as window pane. The window pane internally divided into multiple panes.

Glass pane: This is the first pane closer to the window (screen) and it is used for displaying foreground components.

Content pane: This pane is available behind Glass pane and it is used for display Individual components.

Layered pane: This pane is available behind the content pane and it is used to display Group of components.

Root pane: This pane is available behind the Layered pane and it is used to display back ground components.

Note:

- All the four panes are placed on top of one another and they are transpired.

-All the swing components are available in javax.swing pacakage.

Creation of JFrame

```
import javax.swing.*;
class JFrameDemo{
public static void main(String[] args){
JFrame jf = new JFrame();
jf.setVisible(true);
jf.setSize(400,500);
jf.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        }
}
//JLabel
import javax.swing.*;
import java.awt.*;
```

```
class JFrameDemo extends JFrame{
JLabel jl;
JFrameDemo(){
jl = new Label("Good Moring");
Container c = this.getContentPane();
c.setLayout(new FlowLayout());
c.setBackground(color.black);
Font f = new Font("arial",Font.BoLD,34);
jl.setFont(f);
jl.setForeground(Color.white);
c.add(jl);
this.setVisible(true);
this.setSize(400,400);
this.setTitle("Label");
this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        }
Public Static void main(String[] args){
new JFrameDemo();
     }
}
public static void main(String[] args){
new JFrameDemo();
        }
}
```

JRadioButton: This component allows the user to select only one item from a group items.

Creation of JRadioButton:

```
JRadioButton jrb = new JRadioButton();
JRadioButton jrb = new JRadioButton(Label);
JRadioButton jrb = new JRadioButton(Label,boolean);
```

ButtonGroup: This class is used to place multiple RadioButton into a single group. So that the user can select only one value from that group.

Creation of ButtonGroup:

```
ButtonGroup bg = new ButtonGroup();
```

We can add RadioButtons to the ButtonGroup by using add method.

```
  bg. Add(jrb);
```

CheckBox: This component allows the user to select multiple item from a group of items.

Creation of JCheckBox:

```
JCbeckBox jcb = new JCheckBox();
JCbeckBox jcb = new JCheckBox(Label);
JCbeckBox jcb = new JCheckBox(Label,boolean);
```

JTextField: This component allows the user to type some text in a single line.

Creation of JTextField:

```
 JTextField jtf = new JTextField(size);
```

JTextArea: This component allows the user to type the text in multiple lines.

Creation of JTextArea:

```
JTextArea jta = new JTextArea(rows,cols);
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
class Example extends JFrame implements ActionListener{
JRadioButton eng,doc;
ButtonGroup bg;
JTextField jtf;
JCheckBox bcb,ccb,acb;
JTextArea jta;
Example(){
eng = new JRadioButton("Engineer");
doc = new JRadioButton("Doctor");
bg = new ButtonGroup();
bg.add(eng);
bg.add(doc);
jtf = new JTextField(20);
bcb = new JCheckBox("Bike");
ccb = new JCheckBox("car");
acb = new JcheckBox("aeroplane");
jta = new JTextArea(3,20);
Container c = this.getContentpane();
c.SetLayout(new FlowLayout());
//Registering the listeners with the components
eng.addActionListener(this);
doc.addActionListener(this);
bcd.addActionListener(this);
ccb.addActionListener(this);
acb.addActionListener(this);
c.add(eng);
c.add(doc);
c.add(jft);
c.add(bcd);
c.add(ccb);
c.add(acb);
c.add(jta);
this.setVisible(true);
this.setSize(500,500);
this.setTitle("Selection example");
this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        }
Public void actionPerformed(ActionListener xyz){
if(xyz.getSource() == eng){
jtf.SetText("you are an Engineer");
    }
```

```
if(xyz.getSource() == doc){
jtf.SetText("you are an Doctor");
      }
Strig str = " ";
if(bcb.is Selected()){
str + = "Bike\n";
     }
if(bcb.is Selected()){
str += "car\n";
    }
if(acb.is Selected()){
str += "Aeroplane";
   }
Jta.setText(str);
  }
public static void main(String[] args){
new Example();
       }
}
```

JTable: This component is used to display the data in the form of rows and colums.

Creation of JTable: JTable jt = new JTable(rowData,ColumnNames);

The row data represents a two dimensional array and the columnNames represents a single dimensional array.

Methods of JTable:

1. **getRowCount()**: This method returns the count of the number of rows available in the table.
2. **getCoulmnCount()**: This method returns the count of the number of columns available in the table.
3. **getSelectedRow**: This method returns the index of the row that is selected. It returns -1 when number row is selected
4. **getSelectedRows**: This method returns the indexes of the rows that are selected.
5. **getSelectedRowCount()**: This method returns the count number of rows that are selected.
6. **getSelectedColumn()**: Method returns the index of the column that is selected. It returns -1 if no column is selected.
7. **getSelectedColumn()**: Returns the indexes of the columns that are selected.
8. **getSelectedColumnCount()**: Returns the number of columns that are the Selected.
9. **getValueAt(row,column)**: This method returns a value. That is available in the specified location.
10. **getJTableHeader()**: This method returns the heading of the table.

```
import javax.swing.*;
import javax.swing.table.*;
import java.awt.*;
class JTableDemo extends JFrame{
JTable jt;
JTableDemo(){
String[][] data = {{"abcd","java","70"}, {"defg", "orcle", "80"}, {"xyz",".net","90"}};
String[] names = {"Name", "course", "Marks"};
jt = new JTable(data,names);
JTableHeader head = jt.getTableHeader();
Container c = this.getCotentPane();
c.SetLayout(new BorderLayout());
```

```
c.add("North", head);
c.add("Center",jt);
this.setVisible(true);
this.setSize(300,400);
this.setTitle("JTableDemo");
this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);      }
public static void main(String[] args){
new JTableDemo();
}
      }
```

JprogressBar: This component will display the progress of a task visiually.

Creation of JprogressBar:

JprogressBar pbar = new JprogressBar();

JProgressBar pbar = new JprogressBar(int orientation);

Orientation will specify whether the progressBar should be displayed either horizontally or vertically.

Methods of ProgressBar:

1. **SetMinimum(int)**: This method will set minimum value of the progressbar.
2. **SetMaximum(int)**: Set the maximum value of the progressbar.
3. **SetValue(int)**: This method will set maximum value the progressbar. The value to this method must be with in the range of min and max value.
4. **getMinimum()**: This method returns the minimum value set to the progressbar.
5. **getMaximum()**: The maximum value set to the progressbar.
6. **getValue()**: The current status of the progressbar.
7. **SetOrientation(int)**: This method returns is use to specify the Orientation of the progressbar.
8. **getOrientation()**: This method will return the orientation of the progressbar.
9. **SetStringPainted(boolean)**: This method will display the percentage of the task that is executed.

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
class JProgressBarDemo extends JFrame implements ActionListener{
JProgressBar pbar;
JButton b;
JProgressBarDemo(){
pbar = new JProgressBar();
pbar = setMinimum(0);
pbar = setMaximum(100);
pbar = setValue(0);
pbar = setForeground(color.red);
pbar = setStringpainted(true);
b = new JButton("Click Here");
Container c = this.getContentPane();
c.setLayout(new FlowLayout());
c.add(pbar);
c.add(b);
b.addActionListener(this);
```

```
this.setVisible(true);
this.setSize(400,400);
this.setTitle("JProgressBar");
this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      }

public void ActionPerformed(ActionEvent ae){
if(pbar.getValue() == pbar.getMaximum())
s.exit(0);
pbar.setValue(pbar.getValue(7+5));
      }
public static void main(String[] args){
new JProgressBarDemo();
         }
  }
```

JButton: This component can be used to perform some operation when the user click on it.

Creation of JButton:

```
JButton jb = new JBtton(label);
```

JComboBox: This component will display a group of items as drop down menu from which one of the items can be selected.

Creation of JComboBox:

```
JComboBox jcb = new JComboBox();
```

We can add the items to the ComboBox by using add item method.

```
jcb.addItem(item);
```

Pane: pane is an area in which we can display the components.

JTabbedPane: It is a pane which can contain tabs and each tab can display any component in the same pane.

-To add the tabs to the JTabbedPane we can use the following methods.

```
jtp.add(TabName,Components)
jtp.addTab(TabName,Component)
```

Border: Border is an interface using which we can apply a border to every component. To create the borders we have to use the methods available in BorderFactory calss.

-We can apply the created border to any component by using SetBorder method.

```
Component.SetBorder(Border);
import.java.swing.*;
import.javax.swing.border.*;
import.java.awt.*;
class JTabbedPaneDemo extends JFrame{
JTabbedPane jtp;
JTabbedPaneDemo(){
jtp = new JTabbedPane();
jtp = addTab("Button", new ButtonPanel());
jtp = addTab("ComboBox", new ComboPanel());
Container c = this.getContentPane();
c.SetLayout(new FlowLayout());
```

```
c.add(jtp);
SetVisible(true);
SetSize(400,400);
SetDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        }
public static void main(String args[]){
new JTabbedPaneDemo();
        }
  }
Class Button_Panel extends JPanel{
JButton b1,b2,b3,b4;
ButtonPanel(){
b1 = new JButton("Button1");
b2 = new JButton("Button2");
b3 = new JButton("Button3");
b4 = new JButton("Button4");
Border b = BorderFactroy.createBevelBorder(BevelBorder.LOWERED, color.red, color.green);
        b1.setBorder(b);
b = BorderFactory.create BevelBorder(BevelBorder.LOWERED,color.red,color.green);
        b2.setBorder(b);
b = BorderFactroy.createLineBorder(color.blue,10);
        b3.setBorder(b);
b = BorderFactroy.createMatteBorder(5,10,15,20,color.red);
        b4.setBorder(b);
add(b1);
add(b2);
add(b3);
add(b4);
      }
}
class ComboPanel extends JPanel{
JComboxBox jcb;
ComboPanel(){
jcb = new JComboBox();
jcb.addItem("Hyderabad");
jcb.addItem("Chennai");
jcb.addItem("Delhi");
jcb.addItem("Nellore");
add(jcb);
  }}
```

JMenuBar: This component is used to crate a menu bar which can contain some menus.

Creation of menu bar:

JMenuBar mbar = new JMenuBar();

JMenu: This component is used to create a menu which can contain some menu item.

Creation of JMenu:

JMenu FileMenu = new JMenu("File");

- We can add the menu to the menu bar by using add Method.

mbar.add(FileMenu);

JMenuItem: This component is used to create menu's items which can be placed on to the menu.

Creation of JMenuitem:

JMenuItem newItem = new JMenuItem("New");

-We can add the menu item to the menu by using add method.

FileMenu.add(newItem);

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
class MenuDemo extends JFrame implements ActionListener{
JMenuBar mbar;
JMenu FileMenu, EditMenu;
JMenuItem newItem, openItem, saveItem, exitItem, cutItem, copyItem, pasteItem;
JCheckBoxMenuItem cbox;
MenuDemo(){
mbar = new JMenuBar();
FileMenu = new JMenu("File");
EditMenu = new JMenu("Edit");
mbar.add(FileMenu);
mbar.add(EditMenu);
newItem = new JMenuItem("new");
openItem = new JMenuItem("open");
saveItem = new JMenuItem("save");
exitItem = new JMenuItem("exit");
cutItem = new JMenuItem("cut");
copyItem = new JMenuItem("copy");
pasteItem = new JMenuItem("paste");
cbox = new JCheckBoxMenuItem("choice");
FileMenu.add(newItem);
FileMenu.add(openItem);
FileMenu.add(saveItem);
FileMenu.add(exitItem);
FileMenu.addSeparator();
EditMenu.add(cutItem);
EditMenu.add(copyItem);
EditMenu.add(pasteItem);
EditMenu.add(cbox);
newItem.addActionListener(this);
openItem.addActionListener(this);
saveItem.addActionListener(this);
exitItem.addActionListener(this);
cutItem.addActionListener(this);
copyItem.addActionListener(this);
```

```
pasteItem.addActionListener(this);
Container c = this.getContentPane();
c.setLayout(new BorderLayout());
c.add("North",mbar);
SetVisible(true);
SetSize(400,600);
SetTitle("menu bar");
setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
Public void actionPerformed(ActionEvent ae){
if(newItem.isArmed())
System.out.println("new item clicked");
if(openItem.isArmed())
System.out.println("open item clicked");
if(exitItem.isArmed())
System.out.println("exit item clicked");
if(cutItem.isArmed())
System.out.println("cut item clicked");
      }
public static void main(String[] args){
new MenuDemo();
      }
}
```

Layouts

Layout will specify the format or the order which the components has to be placed on the container.

Layout manager is a class /component that it responsible for arrange the components on the container according to the specified layout.

Different types of layout that can be used are

1. FlowLayout
2. BorderLayout
3. CardLayout
4. GridLayout
5. GridBagLayout

FlowLayout: This Layout will display the components in sequence from left to right, from top to bottom. The components will always be displayed in firstline and in the firsts line is fill these components displayed next line automatically.

Creation of FlowLayout:

```
FlowLayout fl = new FlowLayout();
FlowLayout fl = new FlowLayout(int align);
FlowLayout fl = new FlowLayout(int align, int hgap, int vgap);
```

```
import javax.swing.*;
import java.awt.*;
class LayoutDemo extends JFrame{
JButton b1, b2, b3, b4, b5;
LayoutDemo(){
```

```
b1 = new JButton("Button1");
b2 = new JButton("Button2");
b3 = new JButton("Button3");
b4 = new JButton("Button4");
b5 = new JButton("Button5");
Container c = this.getContentPane();
FlowLayout fl = new FlowLayout(FlowLayout.LEFT,20,30);
c.setLayout(fl);
c.add(b1);
c.add(b2);
c.add(b3);
c.add(b4);
c.add(b5);
SetVisible(true);
SetSize(400,600);
SetTitle("LayoutDemo");
setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
Public static void main(String[] args){
new LayoutDemo();
      }
}
```

BorderLayout: This Layout will display the components along the border of the container. This Layout contains five locations where the component can be displayed. Locations are North, South, East, West and Center(N,S,E,W & C).

Creation of BorderLayout:

```
BorderLayout bl = new BorderLayout(();
BorderLayout bl = new BorderLayout(int vgap, int hgap);
```

```
import javax.swing.*;
import java.awt.*;
class LayoutDemo extends JFrame{
JButton b1, b2, b3, b4, b5;
Layoutdemo(){
b1 = new JButton("Button1");
b2 = new JButton("Button2");
b3 = new JButton("Button3");
b4 = new JButton("Button4");
b5 = new JButton("Button5");
Container c = this.ContentPane();
BorderLayout bl = new BorderLayout(10,20);
c.setLayout(bl);
c.add("North"b1);
c.add("South"b2);
c.add("East"b3);
c.add("West"b4);
c.add("Center"b5);
```

```
setVisible(true);
setSize(400,400);
setTitle("BorderDemo");
setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
public static void main(String[] args){
new LayoutDemo();
    }
}
```

-To add the components in Border Layout we use add method.

```
add("North",Component);
add(Component, BorderLayout, NORTH);
```

CardLayout: A cardLayout represent a stack of cards displayed on a container. At time only one card can be visible and each can contain only one component.

Creation of CardLayout:

```
CardLayout cl = new CardLayout();
CardLayout cl = new CardLayout(int hgap, int vgap);
First(conis);
```

-To add the components in CardLayout we use add method.

```
add("Cardname",Component);
```

-Methods of cardLayout to access athercards:

```
first(Container);
lost(Container);
next(Container);
previous(Container);
show(Container,cardname);
```

```
import javax.Swing.*;
import java.awt.*;
import java.awt.event.*;
class LayoutDemo extends JFrame implements ActionListener{
JButton b1, b2, b3, b4, b5;
CardLayout cl;
Container c;
LayoutDemo(){
b1 = new JButton("Button1");
b2 = new JButton("Button2");
b3 = new JButton("Button3");
b4 = new JButton("Button4");
b5 = new JButton("Button5");
c = this.getContentPane();
cl = new CardLayout(10,20);
c.setLayout(cl);
c.add("card1",b1);
c.add("card2",b2);
```

```
c.add("card3",b3);
b1.addActionListener(this);
b2.addActionListener(this);
b3.addActionListener(this);
setVisible(true);
setSize(400,400);
setTitle("CardLayout");
setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
public void actionPerformed(ActionEvent ae);
cl.next(c);
    }
public static void main(String args[]){
new LayoutDemo();
    }
}
```

GridLayout: Layout will display the components in the format of rows and columns. The container will be divided into table of rows and columna. The intersection of a row and column cell and every cell contain only one component and all the cells are equal size.

Creation of GridLayout:

```
    GridLayout gl = new GridLayout(int rows, int cols);
    GridLayout gl = new GridLayout(int rows, int cols, int vgap, int hgap);
import javax.swing.*;
import java.awt.*;
class LayoutDemo extends JFrame{
JButton b1, b2, b3;
GridLayout gl;
Container c;
LayoutDemo(){
b1 = new JButton("Button1");
b2 = new JButton("Button2");
b3 = new JButton("Button3");
c = this.getContentPane();
gl = new GridLayout(2,3,10,20);
c.setLayout(gl);
c.add(b1);
c.add(b2);
c.add(b3);
SetVisible(true);
SetSize(400,600);
SetTitle("GridLayout");
setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
  }
public static void main(String[] args){
new LayoutDemo();
```

```
    }
}
```

GridBagLayout: This layout is the most efficient layout that can be used for displaying components. In this layout we can specify the location specify the size and etc.

Creation of GridBagLayout:

```
GridBagLayout gbl = new GridBagLayout();
```

-We can specify the location (or) the size with the help of GridBagConstraints.

Creation of GridBagConstraint:

```
GridBagConstraints cons = new GridBagConstraint();
```

1.gridx, gridy: These constraints will specify the location of the cell where the component has to be placed.

2.Gridwith, gridheight: How many cells can be used to display. The component either horizontally or vertically.

default value of gridwidth and gridheight is '1'.

3.ipadx, ipady: These constraints are used to add extra pixels to the component either horizontally or vertically.

4.weightx,weighty: These constraints will specify how much the component must be resized. Either horizontally or vertically, when the component size is smaller then the container (resized).

Fill: This component used to stretch the component either horizontally or vertically, when the component size is smaller then the container.

```
import javax.swing.*;
import java.awt.*;
class LayoutDemo extends JFrame{
JButton b1, b2, b3, b4, b5;
Container c;
GridBagLayout gbl;
GridBagConstraints cons;
LayoutDemo(){
b1 = new JButton("Button1");
b2 = new JButton("Button2");
b3 = new JButton("Button3");
b4 = new JButton("Button4");
b5 = new JButton("Button5");
c = this.getContentPane();
gbl = new GridBagLayout();
c.setLayout(gbl);
cons.Fill = GridBagConstraints();
cons.Weightx = 0.8;
cons.gridx = 0;
cons.gridy = 0;
gbl.setConstraints(b1,cons);
c.add(b1);
cons.gridx = 1;
cons.gridy = 0;
gbl.setConstraints(b2,cons);
c.add(b2);
cons.gridx = 2;
cons.gridy = 0;
```

```
gbl.setConstraints(b3,cons);
c.add(b3);
cons.gridx = 0;
cons.gridy = 1;
cons.gridwidth = 3;
cons.ipady = 100;
gbl.setConstraints(b4,cons);
c.add(b4);
cons.gridx = 1;
cons.gridy = 2;
cons.gridWdith = 2;
cons.ipady = 50;
gbl.setConstraints(b5,cons);
c.add(b5);
SetVisible(true);
SetSize(400,400);
SetTitle("Layout");
setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
public static void main(String[] args){
LayoutDemo ld = new LayoutDemo();
    }
}
```

APPLET

-In java we can have two types of applications.

1.Standalone applications: The applications that are executed in the context of a local machine are called as standalone applications. There applications use w-1 of system resources and the resources are not sharable. There kind of applications contain main() method.

2.Distributed applications: The applications that are executed in the control of a browser are called as distributed applications. The amount of resources required is very minimum and the resources are sharable. These application will not contain main() method. To develop a distributed GUI we use Applet.

Applet: An applet is a java program which register on the server and will be executed in the client.

Life cycle methods:

1. **int()**: This is the 1st method to be executed this method contains the code for initialization. This method will be executed only one time.
2. **start()**: This method will be executed after init() method. This method contains business logic like connecting to files data bases processing the data, generating report etc. this method will be executed multiple times as long as the applet has the control.
3. **stop()**: This method will be executed when the applet looses the control. This method contains a logic for the performing "cleanup activities". This method will be executed multiple times when ever the applet looses the control.
4. **destroy()**: This is the last method to be executed before terminating the applet. This method is used to destroy the applet. This method executed only one time.

Note: when even destroy() method is called before invoking it will call stop() method.

The above methods can be called as life cycle methods or call back methods.

All the life cycle methods are available in class called as applet and they are available as null implementations.

All the four methods are optional mehods.

Note: The applet class is available in "java.applet" package.

```
import java.applet.*;
import java.awt.*;
public class AppletDemo extends Applet{
public void int(){
SetForeground(color.white);
SetBackground(color.black);
SetFont(new Font("arial",Font.ITALIC,34));
System.out.pritln("inside init()");
   }
public void start(){
System.out.println("inside start()");
   }
public void stop(){
System.out.println("inside stop()");
   }
public void destroy(){
System.out.println("inside destroy()");
   }
public void paint(Graphics g){
g.drawString("hello",50,30);
        }
}
```

Compile the program AppletDemo.java, then write

HTML DOCUMENT

```
<html>
<applet code = "AppletDemo" width = "300" height = "300">
</applet>
</html>
```

Save as AppletDemo.html

The execution of above applet can be done in 2 ways.

Way1: open AppletDemo.html in a browser

Way2: execute the AppletDemo.html file by using appletviewer command.

***appletviewer AppletDemo.html**:

We can write the <applet> in an html document or it can be specified in the java file it self.

```
/*
<applet code = "AppletDemo" width = "300" height = "300">
</applet>
*/
```

Compile: javac appletDemo.java

Execute: appletviewer AppletDemo.java (But it not display in browser)

Note: for every applet there should be one <applet> tag.

```
import java.applet.*;
import java.awt.*;
```

```
import java.awt.event.*;
public class FirstApplet extends Applet implements ActionListener{
TextField tf;
Button b;
AppletContext ac;
public void init(){
tf = new TextField(25);
b = new Button("send");
//getting the AppletContext
ac = this.getAppletContext();
add(tf);
add(b);
b.addActionListener(this);
      }
public void actionPerformed(ActionEvent ae){
String str = tf.getText();
Applet a = ac.getApplet("secont");
SecondApplet sa = (SecondApplet)a;
sa.setText(str);
        }
}
//save: FirstApplet.java
import java.applet.*;
import java.awt.*;
public class SecondApplet extends Applet{
String str;
public void init(){
setForeground(color.white);
setBackground(color.green);
setFont(new Font("arial",Font.BOLD,23));
str = "Applet to Applet Communication";
       }
public void setText(String str){
this.str = str;
repaint();
     }
public void paint(Graphics g){
g.drawString(str,50,50);     }
        }
//save: SecondApplet.java
//HTML Document
<html>
<applet code = "FirstApplet" name = "first" width = "200" height = "200">
</applet>
<applet code = "SecondApplet" name = "second" width = "200" height = "200">
```

```
</applet>

</html>
import javax.swing.*;
import java.awt.*;
/*
<applet code = "Running" width = "800" height = "400">
</applet>
*/
public class Running extends JApplet{
public void paint(Graphics g){
Image i = getImage(getCodeBase(),"car.gif");
for(int x = 0; x <=800; x++){
g.drawImage(i,x,o,null);
try{
Thread.Sleep(50);
 }
Catch(Interrupted Exception ie){
ie.printStackTrace();
                    }
               }
          }
   }
<html>
  <applet code = "Running" width = "800" height = "400">
  </applet>
</html>
```