Java means DURGA SOFT..

CORE JAVA

Material

Java™

India's No.1 Software Training Institute

DURGASOFT

www.durgasoft.com Ph: 9246212143 ,8096969696

Object Orientation:

- To design any enterprise application,we have to use a particular programming language.Before selecting a particular programming language,first we have to select a particular type of programming language.

  There are four types of programming languages:

  1)Unstructured Programming languages.

  2)Structured Programming languages.

  3)Object Oriented Programming languages.

  4)Aspect Oriented Programming languages.

1)Unstructured Programming languages:

- These Programming languages are outdated programming languages,they are introduced at the beginning stages of computers.

  Ex: BASIC --->First interpretive programming language.

  FOTRAN- -->First compilative programming language.

- Unstructured Programming languages are not following any structure to design applications,it is not suggestable in application development.
- Unstructured Programming languages are using mnemonic codes like ADD,SUB,MUL,DIV,GOTO...LOAD..to design applications.Basically,mnemonic codes are available in very less number and which may provide very less no. of features to design applications.
- Unstructured Programming languages are using only "goto" statement to define flow of execution,only "goto" statement is not sufficient to define flow of execution.
- Unstructured Programming languages are not allowing functions feature,it may increase code redundancy (duplicate),it will reduce code reusability,it is not suggestable in application development.

2)Structured Programming languages:

- Structured Programming languages are very good when compared with Unstructured Programming languages,because,
- These Programming languages are not outdated programming languages,which are in use at present application development.
- These Programming languages are using high level syntaxes.
- These Programming languages are using more no.of flow controllers to define very good flow of execution.
- These Programming languages are using functions feature to improve code reusability.

Structured Programming languages are not good when compared with Object Oriented Programming languages,because

1)Modularity is not good in Structured Programming languages.

2)Abstraction is not good in Structured Programming languages.

3)Code Reusability is not good in Structured Programming languages.

4)Structured Programming languages are able to provide tightly coupled design for the applications.

5)Security is not good with Structured Programming languages.

3)Object Oriented Programming Languages:

- Object Oriented Programming Languages are very good when compared with Structured Programming Languages,because

  1) It will provide modularity.

  2) It will provide Abstraction.

  3) It will provide Security.

  4) It will provide Code reusability.

  5) It will provide Loosely coupled design.

- Object Oriented Programming Languages are not good when compared with Aspect Oriented Programming Languages,because,

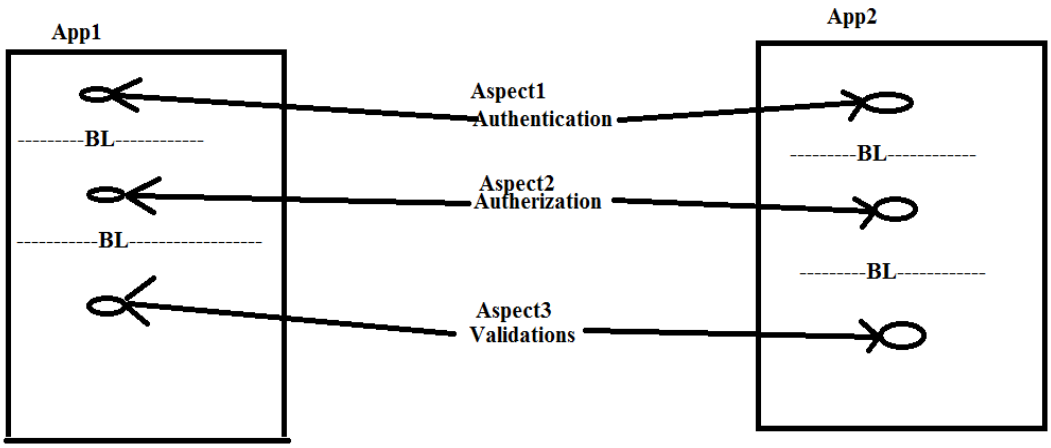
  1) It will not provide very good sharability.

  2) It will not provide very good Reusability.

  3) It will not provide very good loosely coupled design.

4)Aspect Oriented Programming Languages:

- If we design any enterprise application on the basis of Object Oriented then we have to provide the services like Authentication,Authorization,Validation,Logging..along with Business logic in combined manner,it will make compilation,debugging,testing...all the preprocessings are difficult.
- To overcome the above problem,we have to apply Aspect Orientation on Enterprise application.In the case of Aspect Orientation,we have to separate all the services logic form Bussiness Logic,we have to declare each and every service as an aspect and we have to inject these services to the enterprise application at runtime.
- In enterprise application development,Aspect Orientation will provide Loosely Coupled Design,more Sharability and More ReuseAbility.



Note:Aspect Oriented Programming Languages is misterminalogy,no programming language is defined on the basis of Aspect Orientation,Aspect Orientation is a methodology,it is a set of rules and regulations applied on Object Oriented Programming inorder to get more loosely coupled design.

Object Oriented Features:

- To expose the nature of Object Orientation,Object Orientation has provided the following features.
  1)Class
  2)Object
  3)Encapsulation
  4)Abstraction
  5)Inheritance
  6)Polymorphism
  7)Message Passing

- On the basis of above Object Oriented features,there are two types of Programming languages.

  1)Object Oriented Programming Languages
  2)Object Based Programming languages.

Q)What is the difference between Object Oriented Programming languages and Object Based Programming languages?

Ans:

Object Oriented Programming languages will allow allmost all the Object Oriented Features including "Inheritance" Ex:Java	Object Based Programming languages will all allmost all the Object Oriented features excluding "Inheritance". Ex:Java Script

Q)What is the difference between Class and Object?

Ans:

1)Class is a group of elements having similar properties and behaviours.	Object is an individual element among the group of elements having physical identity,physical properties and physical behaviours.
2)Class is virtual.	Object is physical.
3)Class is virtual encapsulation of properties and behaviours.	Object is physical encapsulation of properties and behaviours.
4) Class is generalization.	Object is specialization.
5) Class is a model or blue print for the objects but	Object is an instance of the class.

Q)What is the difference between encapsulation and abstraction?

Ans:Encapsulation is the process of binding data and coding part as a single unit. The process of visualizing necessary implementation or elements and the process of hiding unnecessary elements or implementation part is called as Abstraction.
In Application development,both Encapsulation and Abstraction are achieving "Security".

Inheritance:

- The process of getting variables and methods from one class to another class is called as "Inheritance".
- The main Objective of Inheritance is "Code Reusability".

PolyMorphism:

- PolyMorphism is a greek word,where poly means many and morphism means Structures.
- If one thing is existed in more than one form then it is called as PolyMorphism.

Message Passing:

- The process of bypassing the data along with flow of execution from one instruction to another instruction is called as Message Passing.

Containers in Java:

- Container is an element,which contains some other java elements like varaibles,methods,blocks....

There are three types of containers in java:

1)class

2)Interface

3)Abstract

class:

- The main intention of classes in java programming is to represent the real world entities like Customer,Employee,Student,Account...
- If we want to use classes in java Applications,then we have to use the following steps.

1)Declare a class by using "class" keyword.

2)Declare entity data in the form of variables.

3)Declare entity behaviours in the form of methods.

4)In main class,In main() method,create an object and reference variable for the class.

5)Access the members of the class by using reference variable.

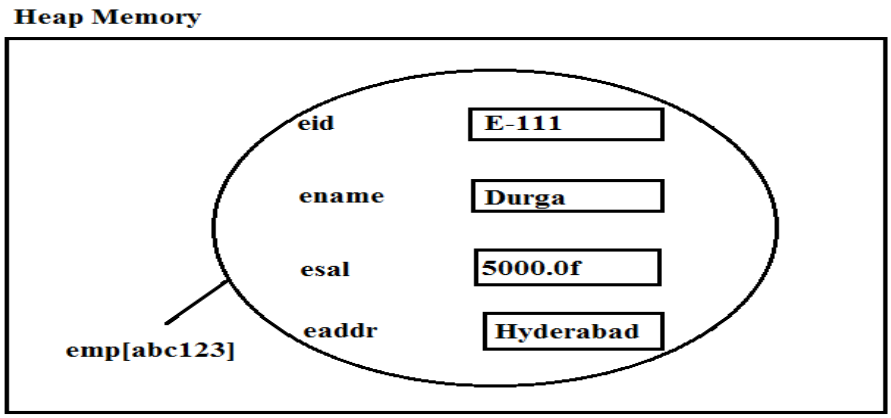
Ex:

```
class Employee{
String eid="E-111";
String ename="Durga";
float esal=50000.0f;
String eaddr="Hyderabad";
void display_Employee(){
System.out.println("Employee Details");
System.out.println("------------------");
System.out.println("Employee Id   :"+eid);
```

```
System.out.println("Employee Name :"+ename);
System.out.println("Employee Salary:"+esal);
System.out.println("Employee Address:"+eaddr);
}}
class Test{
public static void main(String args[]){
Emloyee emp=new Employee();
emp.display_Employee();
}}
```


Output:

```
Employee Details:
-------------------
Employee Id      :E-111
Employee Name: Durga
Employee Salary:50000.0
Employee Address:Hyderabad
```



Ex:

```
class Student{
String sid="S-111";
String sname="Durga";
String saddr="Hyderabad";
int smarks=100;
void display_Student(){
System.out.println("Student Details");
System.out.println("-----------------");
System.out.println("Student  ID:"+sid);
System.out.println("Student name:"+sname);
System.out.println("Student Marks :"+smarks);
System.out.println("Student Address:"+saddr);
}}
```


```
class Test{
public static void main(String[] args){
Student s=new Student();
s.display_Student();
}}
```

Output:

```
Student Details
------------------
Student  ID:S-111
Student  name:Durga
Student  Marks:100
Student  Address:Hyderabad
```

class Syntax:

```
[Access_Modifiers] class class Name[extendsSuper_Class][implements interface_List]
  {
  ----variables------------
```

----constructors---------

----methods------

----blocks-------

----classes------

----abstract classes-----

----interfaces------

}

NOTE: Inside the class,we can write a class,an abstract classes and Interfaces.

Inside the abstract class,we can write a class,an abstract class and an Interface.

Inside the interface,we can write a class,an abstract class and an Interface.

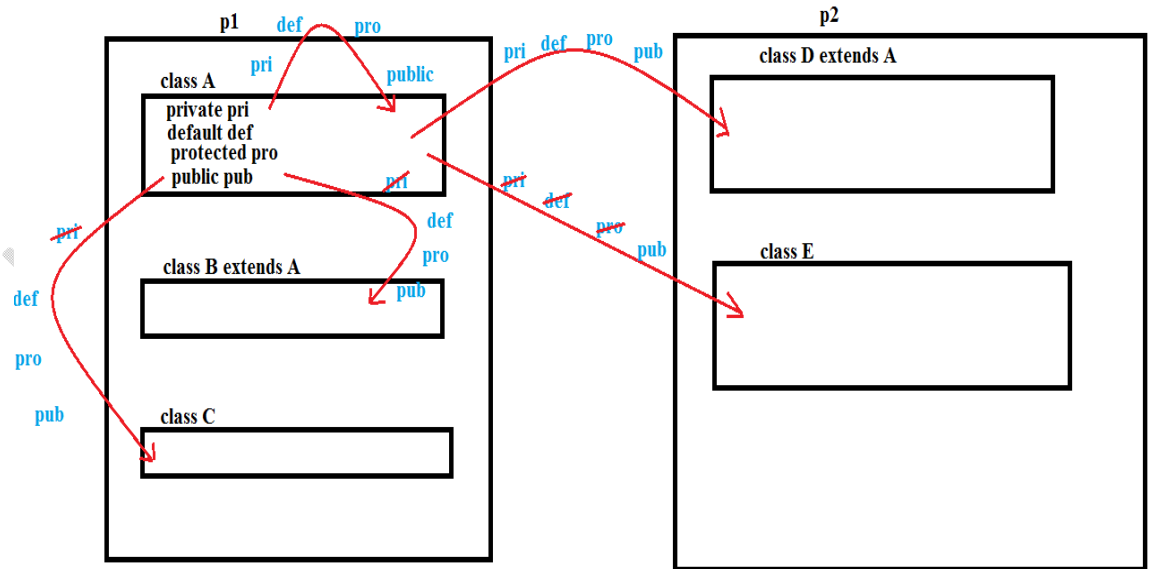
Access_Modifiers:

There are two types of Access Modifiers:

- To define Scope for the java elements like varaibles, methods,classes,.....we have to use the following access modifiers.
  - public

- protected
- default
- private

- Where "private" members will be available upto the present class where we have declared private members.
- Where " <default>" members will be available upto all the classes and interfaces available in the current package.
- Where "protected" members will be available upto all the classes and interfaces available in the present package and the child classes available in the other packages.
- Where "public" members will be available to all the classes and interfaces available in all the packages.



Q)Which access-modifiers are suitable for classes and which access-modifiers are not suitable for classes?

Ans:From the above access modifiers,only public and <default> are allowed for the classes,private and protected are not allowed for the classes.

NOTE:The access modifiers private and protected scopes are defined on the boundaries of classes so we can not apply them for the classes,we can apply them for the members of the classes.

NOTE:All the access modifiers like public,protected,default and private are allowed for the inner classes.

Some Examples on Access Modifiers:

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

Status:No Compilation Error

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

Status:No Compilation Error

==========

```
private class Test{
public static void main(String args[]){
System.out.println("class Test");
}}
```

Status:Compilation Error

Error:modifier private not allowed here.

==========

protected class Test{

public static void main(String args[]){

System.out.println("class Test");

}}

Status :Compilation Error

Error:modifier protected not allowed here

==========

class A{

public class B{

System.out.println("Hai");

}}

Status:No Compilation Error

==========

class A{

private class B{

System.out.println("Hai");

}}

Status:No Compilation Error

==========

class A{

protected class B{

System.out.println("Hai");

}}

Status:No Compilation Error

==========

class A{

}

class B{

System.out.println("Hai");

}}

Status:No Compilation Error.

==========

class A{

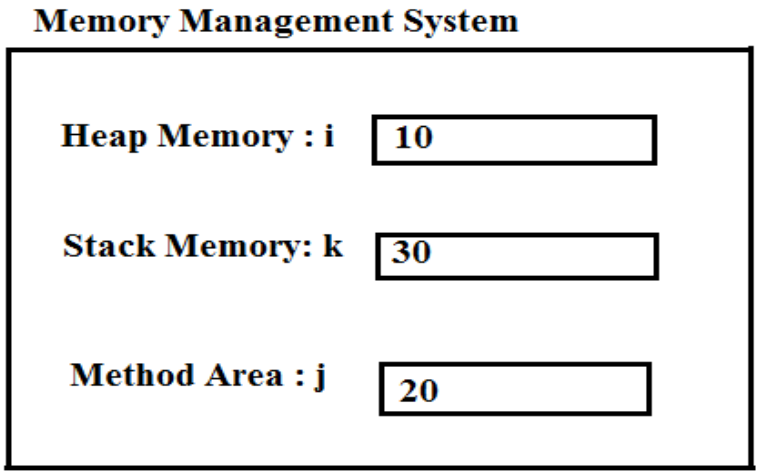
```
}
```

Note:If we don't mention the scope to the class,then it automatically recognize the scope of the class is "default".

Memory Management System:

Ex:

```
Class A{
int  i=10;
static int j=20;
void m1(){
int k=30;
}}
```



All the class variables are called as Instance Variables.

In Java memory management System,it have 3 memories.They are:

1.Heap Memory

2.Stack Memory

3.Method Area

Heap Memory:All the class level variables are stored in Heap Memory.

Stack Memory:All the local variables are stored in Stack Memory.

Method Area:All the static variables are stored in Method Area.

Note:static variables are always declared in the class scope only.

From the second type of access modifiers,which are allowed for classes and which are not allowed for the classes

Ex:

```
static class A{

}
```

```
class Test{
public static void main(String args[]){
System.out.println(“Hai”);
}}
```

Output:

modifier static not allowed here.

==========

Ex:

```
abstract class A{
}
class Test{
public static void main(String args[]){
System.out.println(“Hai”);
}}
```

Output:

No Compilation Error.

==========

Note:static keyword is not allowed for the classes

Note:abstract keyword is allowed for the classes.

Ex:

```
native class A{
}
class Test{
public static void main(String args[]){
System.out.println("Hai");
}}
```

Ouput:modifier native not allowed here.

==========

Note:native keyword is not allowed for the classes.

Ex:

```
final class A{
}
class Test{
```

```
public static void main(String args[]){
System.out.println("Hai");
}}
```

Output:No Compilation Error.

==========

Note:final keyword is allowed for the classes.

Ex:

```
transient class A{

}
class Test{
public static void main(String args[]){
System.out.println("Hai");
}}
```

Output:modifier transient not allowed here

==========

Note: transient keyword is not allowed for the classes.

Note: synchronized modifier is not allowed for the classes.

Note:strictfp modifier is allowed for the classes.

Note:volatile modifier is not allowed for the classes.

Ex:

```
class A{
static int i=10;
int j=20;
}
class Test{
public static void main(String args[]){
        A a=new A();
        System.out.println(a.j);
        System.out.println(A.i);
        System.out.println(a.i);
        }}
```

Output:

20

10

10

If we want to access the non-static members outside of the class, then we must create an object for the class, with the generated reference variable for the class, we are able to access them.

Note:static keyword is allowed for the inner classes.

final keyword is allowed for the inner classes.

abstract keyword is allowed for the inner classes.

native keyword is not allowed for the inner classes.

volatile keyword is not allowed for the inner classes.

transient keyword is not allowed for the inner classes.

synchronized keyword is not allowed for the inner classes.

strictfp keyword is allowed for the inner classes.

Note:All the access modifiers like "public","protected","default"

And "private" are allowed for the inner classes.

- To define some extract nature to the Java elements like variables,methods,classes.....we will use the following access modifiers.

  static

  final

  abstract

  native

  volatile

  transient

  synchronized

  strictfp

- From the above list of access modifiers,classes are able to allow abstract,final and strictfp.

NOTE: Inner classes are able to allow static,final,abstract and strictfp access modifiers from the above list.

Q) Is it possible to declare a class is static or not?

- **Ans**:No,In Java,it is not possible to declare an outer class as static,it is possible to declare an inner class as static,because in Java,static keyword is defined on the basis of classes origin,so that,it can not be applied for the classes,It can be applied for the members of the class including inner classes.

"class" Keyword:

- Where "class" is a keyword,it can be used to represent "class" object oriented feature in Java.

class_Name:

- Where "class_name" is an identifier assigned to the classes to recognize the classes independently.

"extends" keyword:

- Where "extends" is a keyword,it can be used to specify super class in class syntax inorder to reuse the members of super class in the present subclass.

NOTE: In class syntax,extends keyword will not allow more than one super class names,because,if we provide more than one super class names in along with extends keywords then it is becoming as multiple inheritance,it is not possible in Java.

"implements" keyword:

- Where "implements" is a java keyword,which can be used to specify one or more interface names inorder to implement all the interface methods in the present class.

NOTE: In class syntax,extends keyword will allow only one super class but implements keyword will allow more than one interface.

NOTE: In class syntax,both extends and implements keywords are optional,we can write a class with out extends keyword,with out implements keyword and without extends and implements keywords.If we want to write both extends and implements keywords in class syntax then first we have to write extends keyword then we have to write implements keyword.

Ex1:

class A{} --→ Valid

public class A{} --→Valid

private class A{}--→InValid

class A{ private class B{} }-→Valid

class A extends B{ }-→Valid

class A extends B,C{ }-→InValid

class A implements I{ }-→Valid

class A implements I1,I2{ }-→Valid

class A implements I extends B{ }-->Invalid

class A extends B implements I{ }-->Valid

static class A{ }--Invalid

class A{ static class B{ } }--->Valid

Abstract Classes:

- Abstract Classes are Java classes,which will allow both concrete methods and abstract methods.
- In Java applications for Abstract Classes we are able to declare reference variables,we are unable to create Objects.
- If we want to use Abstract classes in java applications,then we have to use the following steps

a)Declare an abstract class by using "abstract" keyword

b)Declare concrete methods and abstract methods as per the application requirement.

c)Declare a sub class for the abstract class.

d)Implement all the abstract methods of abstract class at the sub class,if we voilate this rule then compiler will rise an error.

e)In main class,in main() method,declare reference variable either for abstract class or for sub class but we must create object for sub class.

f)Access abstract class members.

NOTE: If we declare reference variable for abstract class then we are able to access only abstract class members. If we declare reference variable for sub class then we are able to access both abstract class members and sub class members.

Example programme on abstract classes:

```
abstract class A{
void m1(){
System.out.println("m1-A");
}
abstract void m2();
abstract void m3();
}
class B extends A{
void m2(){
System.out.println("m2-B");
}
```

```
void m3(){
System.out.println("m3-B");
}
void m4(){
System.out.println("m4-B");
}}
class Test{
public static void main(String args[]){
    //A a=new A();//Error
    A a=new B();
    a.m1();
    a.m2();
    a.m3();
    //a.m4();--->Error
    B b=new B();
    b.m1();
    b.m2();
    b.m3();
    b.m4();
    }}
```

status:No Compilation Error

Output:

```
m1-A
m2-B
m3-B
m1-A
```

m2-B

m3-B

m4-B

Q)What are the differences between Concrete class and Abstract class?

Ans:

1.Concrete Class is a normal Java class,it will allow 0(zero) or more number of concrete methods.	Abstract class is a Java class,it will allow 0(zero) or more number of concrete methods.
2.In Java applications,for classes only both reference variables and objects.	For abstract classes,we are able to create only reference variables,we are unable to create objects.
3.To declare concrete classes it is not required to use any keyword separately.	To declare abstract classes,we must use "abstract" keyword explicitly.

Interface:

- Interface is a Java container,it able to allow only abstract methods,it will not allow concrete methods.
- Interface is a Java element,it can be utilized declare or expose only service names in the form of abstract methods,it will not provide service implementations.
- In java applications,for the interfaces,we are able to create only reference variables,we are unable to create objects.
- In the case of interfaces,bydefault,all the variables are "public static final".
- In the case of interfaces,by default,all the methods are "public and abstract".

NOTE:While implementing abstract methods,we must declare the methods with public otherwise compiler will rise an error.

Procedure to use Interfaces in Java Applications:

a)Declare an interface by using "interface" keyword.

b)Declare all the service names in the form of abstract methods[Here no need to use "abstract" keyword]

c)Declare an implementation class by including "implements" keyword for the interface[Like sub class in the case of abstract classes but not sub class].

d)Implement all the interface methods at implementation class by declaring "public".

e)In main class,in main() method,declare reference variable either for interface or for implementation class but we must create object for implementation class.

f)Access the interface members.

NOTE:If we declare reference variable for interface then we are able to access only interface members,we are unable to access implementation class own members.

If we declare reference variable for implementation class then we are able to access both interface members and implementation class own members.

Example Programme on Interfaces:

```
interface AccountService{

        void create(String accNo,String accName,String accType);

        void search(String accNo);

        void delete(String accNo);

        }

class AccountServiceImple implements AccountService{
```

```
public void create(String accNo,String accName,String accType){
System.out.println("Account "+accNo+" is created successfully with the following details.");
System.out.println("Account details");
System.out.println("------------------");
System.out.println("Account Number :"+accNo);
System.out.println("Account Name :"+accName);
System.out.println("Account Type :"+accType);
}
public void search(String accNo){
System.out.println("Account "+accNo+" Search Success");
System.out.println("Account details");
System.out.println("----------------");
System.out.println("Account Number  :"+accNo);
System.out.println("Account Name    :Durga");
System.out.println("Account Type    :Savings");
}
public void delete(String accNo){
System.out.println("Account "+accNo+" Deleted Successfully");
}}
class MainProject{
public static void main(String args[]){
AccountService account=new AccountServiceImple();
account.create("abc123","Durga","Savings");
System.out.println();
account.search("abc123");
System.out.println();
```

```
account.delete("abc123");
}}
```

Output:

```
Account abc123 is created successfully with the following details.
Account details
----------------
Account Number :abc123
Account Name : Durga
Account Type :Savings
Account abc123 Search Success
Account details
-----------------
Account Number :abc123
Account Name    :Durga
Account Type     :Savings
Account abc123 Deleted Successfully
```


Q)What are the differences between classes,abstract classes and interfaces?

Ans:

1.Classes will allow only concrete methods.	Abstract classes will allow both concrete methods and abstract methods	Interfaces will allow only abstract methods.
2.For classes only,we are able to create both reference variables and objects.	For abstract classes,we are able to create only reference variables,we are unable to create objects.	For interfaces,we are able to create only referencevariables,we are unable to create objects.
3. No default cases for the variables in the case of classes.	No default cases for the variables in the case of abstract classes.	In the case of interfaces,by default,all the variables are "public static final".
4. No default cases for the methods in the case of classes.	No default cases for the methods in the case of abstract classes.	In the case of interfaces,by default ,all the methods are "public and abstract".
5. Constructors are possible in classes.	Constructors are possible in abstract classes.	Constructors are not possible in interfaces.
6. Classes will provide less share ability.	Abstract classes will provide middle level share ability.	Interfaces will provide more share ability.

Methods in Java:

- In Java applications, the main intention of providing methods is to represent entities actions.
- To provide methods in java applications, we have to use the following syntax.

Syntax:

```
[Access_Modifiers] return_Type method_Name([param_List]) [throws Exception_List]
{
        ---body/implementation---
}
```


- Where Java methods are able to allow the **access modifiers** like public,protected,<default> and private one at a time.
- Where Java methods are able to allow the **access modifiers** like static,final,abstract,native,synchronized,strictfp.
- From the above list of **access modifiers** java methods are able to allow more than one access modifier but they must be available in valid combination.

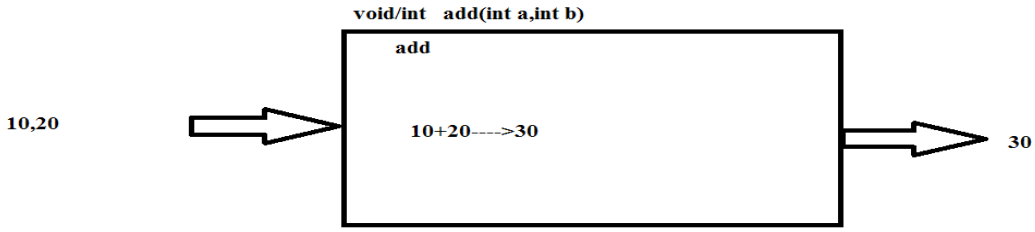
  Ex:static and final are valid combination

  static and abstract are invalid combination

final and abstract are invalid combination

static and synchronized valid combination

- The main intention of providing **return type** to the methods is to specify which type of data we are returning from methods.
- Java is strictly typed programming language, where in java applications before representing data first we have to decide which type of data we are representing. In the case of methods, before returning data first we have to decide which type of data we are returning, for this we must require return type for the methods.
- For the methods, we are able to use all **primitive data types** like byte,short,int,long,float,double,char,boolean and all user defined data types like arrays,classes,abstract classes,interfaces and "void" as return types.
- Where **"void"** return type is representing that the method is not returning any value.



Where **Method_Name** is an identifier,it will be assigned to the methods inorder to recognize the methods independently.

Where the main intention of **parameter list** is to provide some input data in order to perform an action.

In Java methods,we are able to allow all primitive data types,all user defined data types as parameters.

Note:"void" is only return type,it should not be used as parameter to the methods.

where **"throws"** can be used to bypass the generated exception from present method to caller method.

In method syntax, **"throws"** keyword is able to allow more than one exception name.

With respect to the Object state insertion and retrieval there are two types of methods in Java.

1.Mutator Methods

2.Accessor Methods

Q)What is the difference between Mutator Methods and Accessor Methods?

Ans:

1.Mutator Method is a normal Java method,it can be used to set/modify data on an object.	Accessor Method is a normal Java method,it can be used to get the data from an Object.
2. Ex: All setter methods[setXXX(-)} methods in Java Bean classes are treated as Mutator Methods.	Ex: All getter methods[getXXX()] in Java bean classes are treated as Accessor methods.

NOTE:Java Bean class is a normal Java class having variables and the respective setXXX(-) methods and getXXX() methods.

Example Programme on Mutator Methods and Accessor Methods:

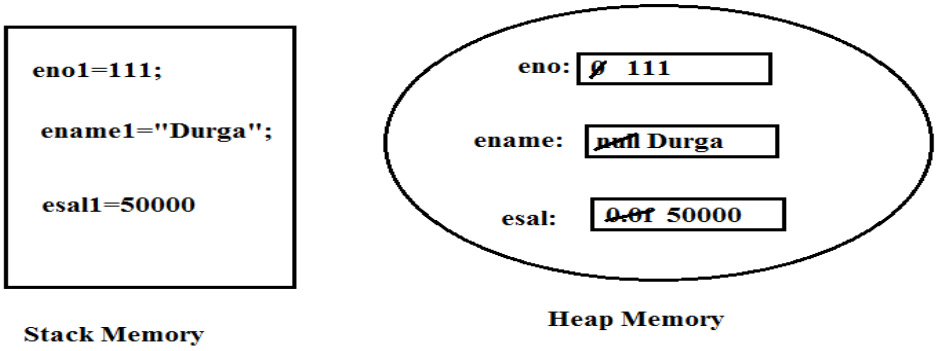
```
class Employee  //Java Bean class
{
private int eno;
private String ename;
private float esal;
public void setEno(int eno1)//Mutator Method
```

```
{
eno=eno1;
}
public void setEname(String ename1)//Mutator Method
{
ename=ename1;
}
public void setEsal(float esal1)//Mutator Method
{
esal=esal1;
}

public int getEno()//Accessor Method
{
return eno;
}
public String getEname()//Accessor Method
{
return ename;
}
public float getEsal()//Accessor Method
{
return esal;
}}
class Test{
public static void main(String[] args){
```

```
Employee e=new Employee();
e.setEno(111);
e.setEname("Durga");
e.setEsal(5000.0f);
System.out.println("Employee Details");
System.out.println("--------------------");
System.out.println("Employee Number  :"+e.getEno());
System.out.println("Employee Name    :"+e.getEname());
System.out.println("Employee Salary    :"+e.getEsal());
}}
```


Output:

```
Employee Details
------------------------
Employee Number  :111
Employee Name    :Durga
Employee Salary    :5000.0
```



Variable-Argument method[Var-Arg Method]:

- In general,in Java applications,if we define any method with "n" no of parameters then to access that method we must pass the same "n" no.of parameters values otherwise compiler will rise an error.
- As per the requirement,we want to define a method and we want to access that method by passing **variable no.of parameters** values.To achive this requirement,**JDK5.0** version has provided a separate method convention called as **"variable-Argument"** method.
- Variable-Argument method is a normal java method,which contains variable-Argument parameter,**where Variable-Argument parameter is an array representation**,it able to store all the argument values in the form of an array while accessing variable-argument method with a parameter values set.

Syntax:

```
return_Type method_Name(Data_Type ... var_Name)
{
----impl-----
}
```

Example Programme on Variable-Argument Method:

```
class A{
void add(int ...a)//int[] a
{
System.out.println("No of Arguments :"+a.length);
```

```
    System.out.println("Arguments List   :");
    int result=0;
    for(int i=0;i<a.length;i++){
    System.out.println(a[i]+" ");
    result=result+a[i];
    }
        System.out.println();
        System.out.println("Addition    :"+result);
        System.out.println("-------------------------");
  }}
class Test{
public static void main(String args[]){
A a=new A();
a.add();
a.add(10);
a.add(10,20);
a.add(10,20,30);
}}
```


Output:

```
No of Arguments :0
Argument List    :
Addition              :0
-----------------------------------
No of Arguments :1
Argument List    :10
Addition          :10
-----------------------------------

No of Arguments :2
Argument List    :10 20
Addition          :30
-----------------------------------
No of Arguments :3
Argument List    :10 20 30
Addition          :60
-----------------------------------
```

- In the case of var-arg method,we are able to pass normal parameters along with var-arg parameter but they must be **provided before var-arg parameter** because in var-arg methods var-arg parameter is always last parameter.
- Due to the above reason,it is not possible to provide more than one var-arg parameter to a single var-arg method.

Ex:

```
void m1(int ... a){}  --->Valid
```

void m1(float f,int ... a){} ---> Valid

void m1(int ... a,float f){} ---> InValid

void m1(int ... a,float ... f){} --->InValid

Syntax to create Object:

- In Java applications,the main intention to create objects is,

  1.To store entities data.

  2.To access the members of a particular class.

- To create an object for a particular class,we have to use the following syntax.

  Class_Name ref_var=new Class_Name([param_List]);

  | |

  ==========

  |

  Constructor

  class A{

  }

  A a=new A();

When JVM encounter the above instruction,JVM will perform the following actions.

1.creating memory

2.Generating Identities

3.Providing initializations\

1.Creating Memory:

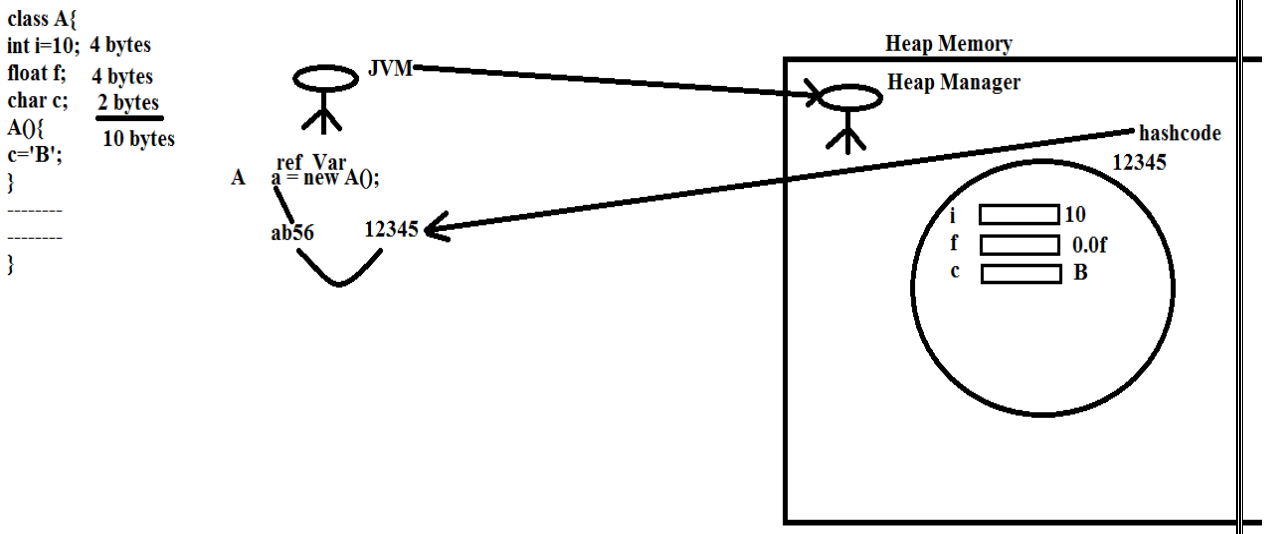
- JVM will send a request to Heap manager to create an object when it encounter **"new"** keyword.
- JVM will identify object memory size by recognizing all the instance variables available in the respective class.
- As per JVM requirement,Heap Manager will create a block of memory at **Heap Memory**.

2.Generating Identities:

- After creating a block of memory for JVM requirements,heap manager will assign a random integer value as an identity for the object called as **"HashCode"** value.
- After getting HashCode value,Heap Manager will send that hashCode value to JVM,where JVM will convert that hashCode value in the form of its Hexa Decimal value called as **"Reference Value".**
- After getting Object reference value,JVM will assign that reference value to a variable called as **"Reference Variable".**

3.Providing initializations:

- After creating object and its identities,JVM will allocate memory blocks of the instance variables in the object on the basis of their data types.
- After getting memory blocks for the instance variables,JVM will provide inital values to the instance variables by searching initializations at the **respective constructor and at the class level declarations**.
- If instance variables are not having initializations at both constructor and at class level declarations then JVM will provide **default values as initial values on the basis of their datatypes**.



To get hashcode value of an object we have to use the following method.

public native int hashcode()

To get reference value of an object,we have to use the following method.

public String toString()

Example programme on hashCode() and toString():

```
class A extends Object{  //Super Class for any Java Class(Object)
int i=10;
float f=22.22f;
}
```

```
class Test{
public static void main(String args[]){
A a=new A();
int hashCode=a.hashCode();
System.out.println(hashCode);
String ref_Value=a.toString();
System.out.println(ref_Value);
}}
```

Output:

```
1802307482
A@6b6d079a
```

- In Java Programming language there is a common and default super class for each and every java class[Predefined classes,user defined classes] that is **"java.lang.Object"** class.

java.lang.Object class contains the following methods.

hashCode()

toString()

equals()

clone()

finalize()

wait()

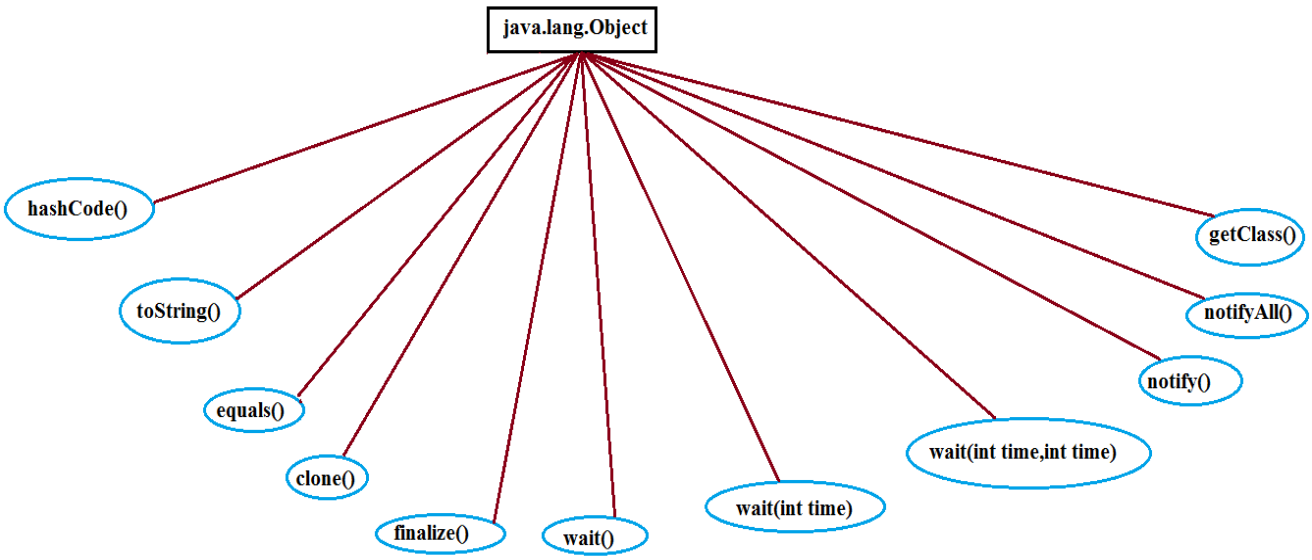
wait(int time)

wait(int time,int time)

notify()

notifyAll()

getClass()



- All the above methods are common to each and every class,we can access these methods on any class reference variable.
- In the above programme,when we access **hashCode()** method and **toString()** method on "class A" reference variable then JVM will search for the respective method first on the respective class, if these methods are not available at the respective class then JVM will search for these methods in the respective super class.If no super class is existed explicitly then JVM will search for these methods in the common and default super class that is **"Object"** class.

Q)In Java applications,every class having a default super class that is "java.lang.Object" class.In Java applications,if we declare a "class B" and if we extend this from "class A" explicitly then "class B" is having two super classes [class A,Object],it shows multiple inheritance clearly then how can we say multiple inheritance is not possible in Java?

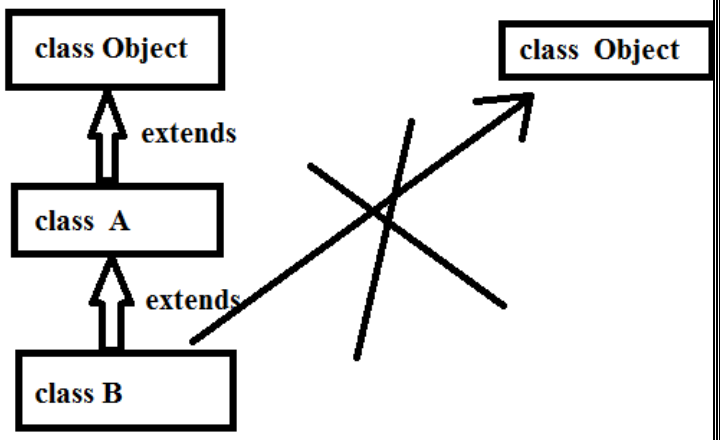
Ans:

```
class A{

}

class B extends A,Object{

}
```


- Ans:In Java applications,Object class is a common and default super class for every java class when ever they are not extending from any super class explicitly.If any class is extending super class then the respective subclass is not having Object class as a super class directly,
- Object class is a super class for the respective sub class through the explicit super class that is through **"multi level"** inheritance not through multiple inheritance.

```
class A extends Object{

}

class B extends A{

}
```

```
class A{
}
class B extends A
{

}
```



In Java applications,if we pass any reference variable as parameter to System.out.println() method then JVM will access toString() method on the provided reference variable.

In this case,JVM will search for toString() method in the respective class,if it is not available then JVM will search for toString() method in explicit super class,still if toString() method is not available then JVM will search and execute toString() method in Object class.

Example Programme on toString() method:

class A

```
{


}
class Test
{
public static void main(String args[])
{
A a =new A();
String ref_Val=a.toString();
System.out.println(ref_Val);
System.out.println(a.toString());
System.out.println(a);
}
}
```

Output:

```
A@546da8eb
A@546da8eb
A@546da8eb
```


In Java applications,any reference variable as parameter to System.out.println(-) method then **JVM will access Object class toString()** method,it will display object reference value,but,as per the application requirement,we want to display the respective entity details instead of object reference value.

To achieve the above requirement,we have **to define our own toString()** method in the respective entity class.

In this case JVM will **execute our own toString()** method instead of executing Object class toString() method.

Example programme on User-Defined toString() method:

```
class Employee{

String eid="E-111";

String ename="Durga";

String eaddr="Hyderabad";

public String toString(){

System.out.println("Employee Details");

System.out.println("------------------");

System.out.println("Employee Id :"+eid);

System.out.println("Employee Name: "+ename);

System.out.println("Employee Address: "+eaddr);

return "";

}}
```



```
class Test{
```

```
public static void main(String args[]){
Employee emp=new Employee();
System.out.println(emp);
}}
```

Output:

```
Employee Details
------------------------
Employee Id  :E-111
Employee Name : Durga
Employee Address : Hyderabad
```

NOTE: In Java some predefined classes like String,StringBuffer,Exception,Thread, all wrapper classes..are not depending on Object class toString() method,they are having their own toString() methods inorder to display their own data instead of Object reference value.

Example programme on toString() method on Predefined Classes:

```
class Test{
public static void main(String args[]){
String str=new String("abc");
System.out.println(str);
```

```
Exception e=new Exception("My Exception");
System.out.println(e);
Thread t=new Thread();
System.out.println(t);
Integer i=new Integer(10);
System.out.println(i);
}}
```

Output:

```
abc
java.lang.Exception: My Exception
Thread[Thread-0,5,main]
10
```

There are two types of Objects in Java:

1.Immutable Objects

2.Mutable Objects

Q)What is the difference between Immutable Object and Mutable Object?

(or)

Q)What is the difference between String and StringBuffer?

Ans:Immutable Object is a Java object,it will not allow modifications on its content,if we perform operations over immutable objects content,data will be allowed for the operations but the resultent data must be stored by creating new object,not in original object.

Ex: String class objects are immutable

All wrapper classes like Byte,Integer,Float,....are immutable

Mutable Object is a Java object,it will allow modifications over the data directly.

Ex:All Java Objects including StringBuffer by default mutable.

```
class Test{
public static void main(String args[]){
String str1=new String("Durga");
String str2=str1.concat("Software");
String str3=str2.concat("Solutions");
System.out.println(str1);
System.out.println(str2);
```

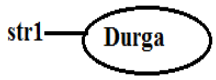
```
System.out.println(str3);
StringBuffer sb1=new StringBuffer("Durga");
StringBuffer sb2=sb1.append("Software");
StringBuffer sb3=sb2.append("Solutions");
System.out.println(sb1);
System.out.println(sb2);
System.out.println(sb3);
}}
```

Output:

```
Durga
DurgaSoftware
DurgaSoftwareSolutions
DurgaSoftwareSolutions
DurgaSoftwareSolutions
DurgaSoftwareSolutions
```

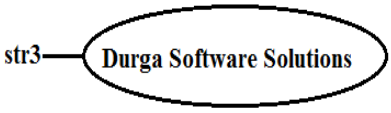
Immutable Objects

```
String str1=new String("Durga");
```



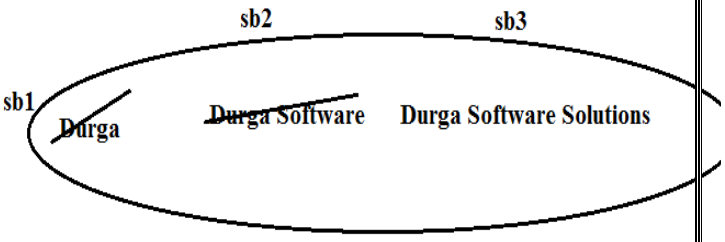

```
String str2=str1.concat("Software");
```


```
String str3=str2.concat("Soluions");
```



Mutable Objects

```
StringBuffer sb1=new StringBuffer("Durga");
StringBuffer sb2=new StringBuffer("Software");
StringBuffer sb3=new StringBuffer("Solutions");
System.out.println(sb1);—Durga Software Solutions
System.out.println(sb2);—Durga Software Solutions
System.out.println(sb3); — Durga Software Solutions
```



Q)What is the difference between Object and Instance?

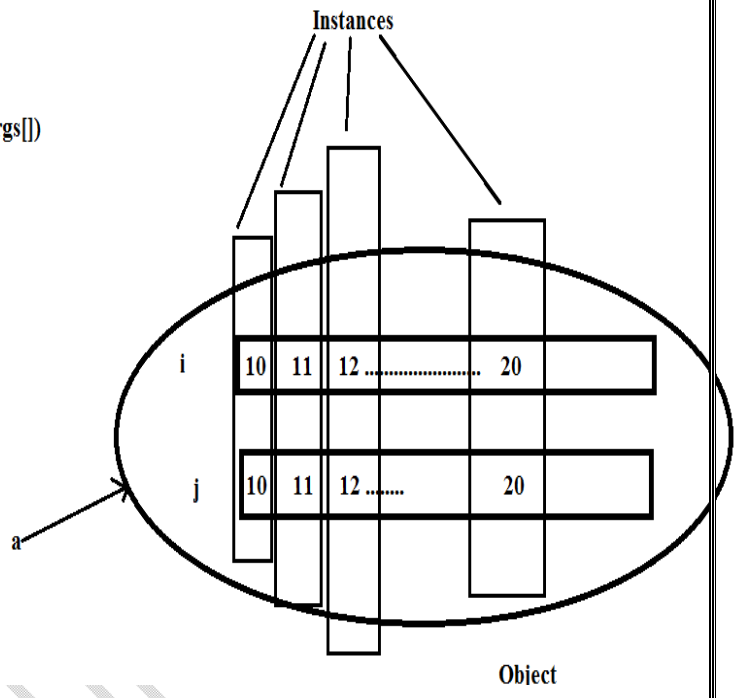
Ans:Object is a memory element to store data.

Instance is a copy of values available in an object at a particular point of time.

Every object is able to manage no. of instances but we are able to access only the latest instance.

```
class A
{
int i=10;
int j=10;
}
```

```
class Test
{
public static void main(STring args[])
{
A a =new A();
for(int x=0;x<10;x++)
{
a.i=a.i+1;
a.j=a.j+1;

}
}
```



Constructors:

1)Constructor is a Java feature,it can be used to **construct the objects.**

2)The role of the constructors in Object creation is to **identify memory size** and to **provide initializations** for the object.

3)Constructors will be executed exactly at the time of creation objects,not before creating object,**not after creating objects.**

4)Constructors will be utilized to provide initializations for the **class level variables** at the time of creating Object.

5)Constructors should have the **same name of class name**.

6)Constructors will not have **return types.**

7)Constructors will not allow the access modifiers like **static,final,abstract**,...

8)Constructors will allow the access modifiers like **public,protected,<default> and private**.

9)Constructors will allow **"throws"** keyword as part of its syntax to bypass the generated exception from present Constructor to caller.

Syntax of Constructors:

```
[Access_Modifier] Class_Name([param_List])[throws Exception_List]
{
-----body------------
}
```

NOTE1:If we declare a constructor with out its class name then compiler will not treate the provided constructor as a constructor,compiler will treate the provided constructor as normal Java method with out the return type,compiler will rise an error like

"Invalid method declaration; return type required".

NOTE2:If we provide return type to the constructor then compiler and JVM will treate that constructor as a normal java method,which satisfies method Syntax.In this case compiler will not rise any error and JVM will not rise any exception.If we access the provided constructor as normal java method then it will be executed.

NOTE3:Java methods may allow the same class name as method name.

Example programme on,classname same as methodname:

```
class A{
void A(){
System.out.println("A-con");
}}
class Test{
public static void main(String args[]){
A a =new A();
a.A();
}}
```

Output:

A-con

NOTE4:If we declared the constructors to the access modifiers like static,final,abstract...then compiler will rise an error like "modifier XXX is not allowed here".

NOTE:If we declare any constructor as private then we have to access that constructor with in the same class,it is not possible to access that constructor in outside of the respective class.

In Java,there are two types of constructors.

1)Default Constructors

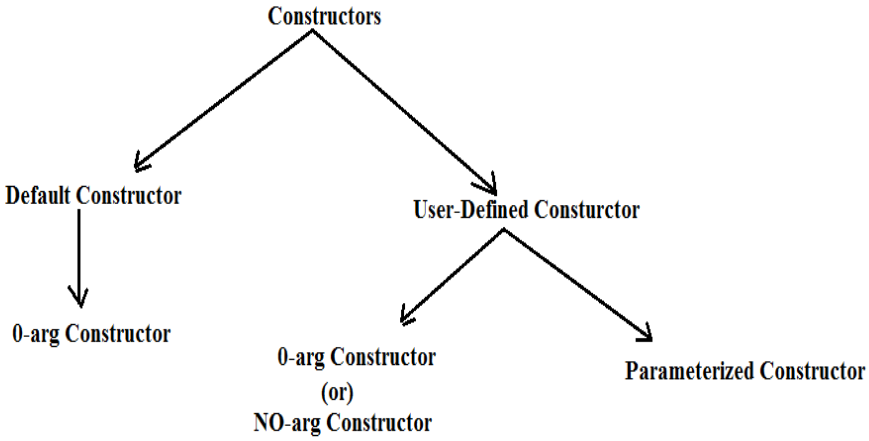
2)User-defined Constructors

If we provide any constructor explicitly in any class then that constructor is called as "User-Defined Constructor".

There are two types of user-defined Constructors

1.0-argument Constructor:It is a constructor with out parameters provided by the developers as per their application requirements.

2.Parametrized Constructor:It is a constructor with at least one parameter provided by the developers as per their application requirements.



Example programme on parameterized constructor:

```
class Account{
String accNo;
String accName;
String accType;
public Account(String accNo1,String accName1,String accType1){
accNo=accNo1;
accName=accName1;
accType=accType1;
}
```

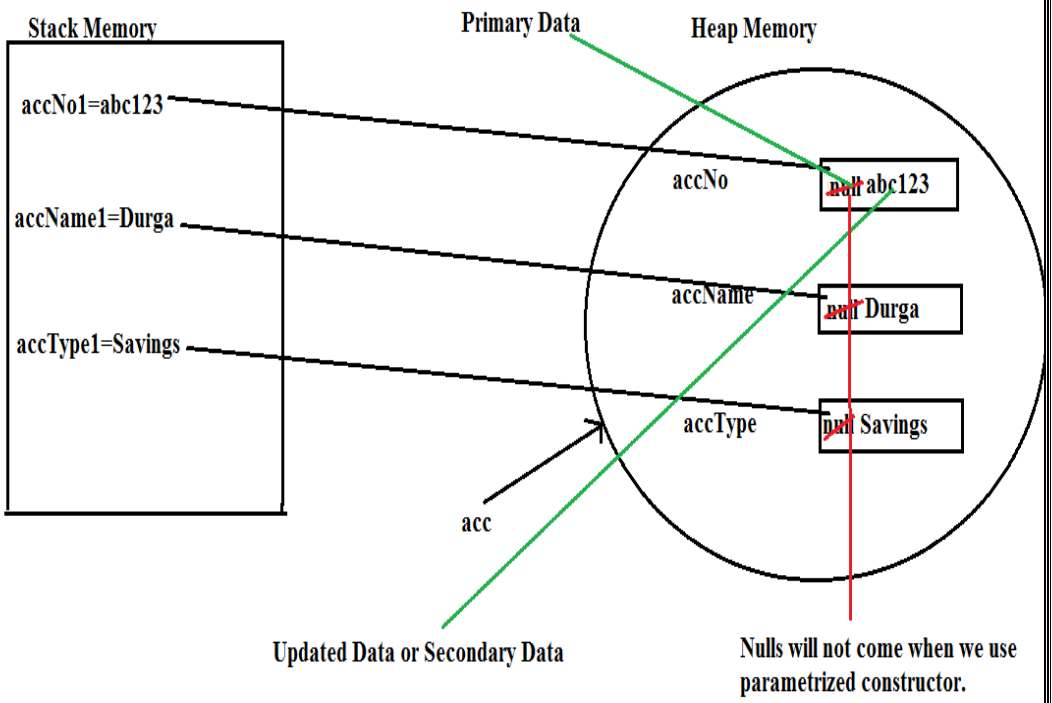


```
public String toString(){
System.out.println("Account is created Successfully with the following details");
System.out.println("Account details");
System.out.println("--------------------");
System.out.println("Account Number :"+accNo);
System.out.println("Account Name :"+accName);
System.out.println("Account Type :"+accType);
return "";
```

```
}}
class Test{
public static void main(String args[]){
Account acc=new Account("abc123","Durga","Savings");
System.out.println(acc);
}}
```

Output:

```
Account is created Successfully with the following details
Account details
------------------------
Account Number : abc123
Account Name    :Durga
Account Type     :Savings
```



In the above programme,if we set account details by using any method like setAccountDetails() then account details will be set to Account class Object after creating Object as secondary data,not as initial data,here in account object initial data will be null values [default values of String].

In the above context,if we want to set our data as initial data in the object then we have to set account details at the time of creating object,not after creating object.To achieve this,we have to use constructor to set data to Object.

Default Constructor:

- It is a 0-argument constructor provided by the **compiler** when ever we are not provided any constructor explicitly.
- Compiler will provide **0-argument constructor** always,compiler will not provide parametrized constructor as default constructor.

NOTE: In Java,default constructors is provided by the compiler only,not by JVM.

D:\JAVA7\Test.java

```
class Test{

}
```

On Command prompt.

D:\JAVA7\javac Test.java

D:\JAVA7\javap Test

compiler from Test.java

```
class Test extends java.lang.Object{

Test();------default constructor

}
```


NOTE: In Java,always default constructors are 0-argument constructors but all the 0-argument constructor need not be default constructors,some 0-argument

constructors are provided by compiler and some other 0-argument constructors are provided by user explicitly.

- If we provide more than one constructor with the same name and with different parameter list then it is called as **"Constructor Overloading".**

Example programme on Constructor Overloading:

```
class Person{

Person(){

System.out.println("0-argument Con");

}
```



```
Person(String name){

System.out.println("String-Parameterized Constructor");

}

Person(String name,String address){

System.out.println("String,String-parameterized con");

}}

class Test{

public static void main(String args[]){

Person p1=new Person();

Person p2=new Person("Durga");

Person p3=new Person("Durga","Hyderabad");

}}
```

Output:

0-argument Con

String -Parameterized Constructor

String,String-parameterized con

Q)Is it possible to overload the constructors or not?

Ans:Yes,it is possible to overload the constructors in Java,but it is not possible to override the constructors in Java because constructors are not inheritable from super class to subclass.

Realtime utilization of Constructors:

Consider the following Java class,

```
class EmployeeService{
void add(String eid,String ename,String eaddr){
        ---load and register the driver----
        ---establish connection-------
        ---create Statement----
        ---execute insert sql query---
}
```



```
void search(String eid){
```

```
        ---load and register the driver----
        ---establish connection-------
        ---create Statement----
        ---execute insert sql query---
}
void delete(String eid){
        ---load and register the driver----
        ---establish connection-------
        ---create Statement----
        ---execute insert sql query---
}}
```

NOTE: If we use the above convention in java applications then JVM has to execute the instructions related to "load and register driver","establish connection","Create Statement" repeatedly for each and every method call like add(),search() and delete() method,it is unnecessary execution,it will increase Connection objects and Statement Objects unnecessarly and it will increase overall application execution time,it may increase overall application execution time,it may reduce application performance.

- To overcome the above problems,we have to use constructors inorder to provide and execute the commonly used code at the time of creating object.

```
class EmployeeService {
EmployeeService{
```

```
        ---load and register the driver----
        ---establish connection-------
        ---create Statement----
}
void add(String eid,String ename,String eaddr){
        ---execute insert sql query---
}
void search(String eid){
        ---execute insert sql query---
}
void delete(String eid){
        ---execute insert sql query---
}}
```


NOTE: In Java,we can utilize constructors to execute some instructions related to the resources creation like Streams creation,FilesCreation,Connection and Statement Objects creation...at the time of creating object.

Instance Context/Instance Flow of execution:

In Java,instance context is represented in the form of the following elements.

1.Instance Variables

2.Instance Methods

3.Instance Blocks

- In Java,for every Object a separate instance context will be created.

Instance Variables:

- Instance Variables is a normal Java variable,whose values will be varied from **one instance to another instance of an object.**
- Instance Variable is a variable,which will be recognized and initialized just **before executing** the respective class constructor.
- In Java applications,instance variables must be declared at class level and non-static, instance variables **never be declared as local variables** and static variables.
- In Java applications,instance variables data will be stored in Object memory that is **"Heap Memory".**
-

2.Instance Methods:

- Instance Method is a **normal Java method**,it is a set of instructions,it will represent an action of an entity.
- In Java applications,instance methods will be executed when we access that method.In Java applications,all the methods wont be executed with out the proper method call.

```
class Account{

public void add(String accNo,String accName,String accType){

}

public void search(String accNo){

}}
```

where add(),search() methods are instance methods.

Example programmes on Instance variables are executed before executing the constructors:

```
class A{
 int i=m1();
 A(){
 System.out.println("A-Con");
  }
 int m1(){
 System.out.println("M1-A");
 return 10;
 }}
 class Test{
 public static void main(String args[]){
 A a =new A();
 }}
```

Output:

M1-A

A-con

```
class A {
   int j=m1();
   int m2(){
   System.out.println("m2-A");
   return 10;
   }
   A(){
   System.out.println("A-con");
```

```
}
 int m1(){
 System.out.println("m1-A");
 return 20;
 }
 int i=m2();
}
class Test{
public static void main(String args[]){
A a =new A();
System.out.println(a.i);
System.out.println(a.j);
a.m1();
a.m2();
}}
```

Output: m1-A
m2-A
A-con
10
20
m1-A
m2-A

Instance Block:

- Instance Block is a set of instructions which will be recognized and executed just **before executing** the respective class constructors.

- Instance Block as are having the **same power of constructors**,it can be used to include the code related to resource creation like Streams,Files,Database Connections....

Syntax:

```
{
 ---instructions----
 }
```

Example programme on Instance Block:

```
class A {
 A(){
 System.out.println("A-CON");
  }
 {
 System.out.println("IB-A");
 }}

 class Test{
 public static void main(String args[]){
 A a=new A();
 } }
```

Output:

IB-A

A-CON

Example programmes on Instance variables,blocks and methods:

```
class A {
 A(){
 System.out.println("A-CON");
 }
 {
 System.out.println("IB-A");
 }
 int m1(){
 System.out.println("m1-A");
 return 10;
 }
 int i=m1();
 }

 class Test{
 public static void main(String args[]){
  A a=new A();
 }}
```

Output:

IB-A

m1-A

A-CON

```
class A{
 int m1(){
```

```
System.out.println("m1-A");
return 10;
}
{
System.out.println("IB-A");
}
int i=m2();
A(){
System.out.println("A-con");
}
int i=m1();
{
System.out.println("IB1-A");
}
int m2();
{
System.out.println("m2-A");
return 20;
}}

class Test{
public static void main(String args[]){
A a1=new A();
A a2=new A();
} }
```

Output:

IB-A

m2-A

m1-A

IB1-A

A-con

IB-A

m2-A

m1-A

IB1-A

A-con

'this' keyword:

- 'this' is a Java keyword,it can be used to represent current class object.

In Java applications,we are able to utilize 'this' keyword in the following four ways.

1.To refer current class variable

2.To refer current class methods

3.To refer current class constructors

4.To refer current class object

1.To refer current class variables:

- If we want to refer current class variables by using 'this' keyword then we have to use the following syntax.

  this.var_Name

- **NOTE: In Java applications,if we have the same set of variables at local and at class level and if we access that variables then JVM will give first priority for local variables,if local variables are not available thenJVM will search for that variables at class level,even at class level also that variables are not available then JVM will search at super class level.At all the above locations,if the specified variables are not available then compiler will rise an error.**
- **NOTE: In Java applications,if we have same set of variables at local and at class level then to access class level variables over local variables we have to use 'this' keyword.**

  Example programme To refer current class variables:

```
class A{

 int i=10;
```

```
int j=20;
A(int i,int j) {
System.out.println(i+"  "+j);
System.out.println(this.i+"  "+this.j);
}}
class Test{
public static void main(String args[]){
A a=new A(100,200);
}}
```

Output:

```
100    200
10     20
```

NOTE:Real time utilization of 'this' keyword while accessing class level variables.In general,in enterprise applications,we are able to write no. of Java bean classes as per application requirement.In Java bean classes,we are able to provide variables and their setter methods and getter methods.In Java bean classes,in setter methods,always we have to assign local variable value to class level variable.In this context,to refer class level variable over local variable we have to use 'this' keyword.In getter methods,always,we have to return class level variables but where it is not required to use 'this' keyword,because,in getter methods no local variables.

Example programme on “this” keyword:

```
class User{
private String uname;
private String upwd;
public void setUname(String uname) {
this.uname=uname;
}
```

```
public void setUpwd(String upwd) {
this.upwd=upwd;
}
public getUname() {
return uname;
}
public getUpwd(){
}
```


2.To refer current class method:

- If we want to refer current class method by using 'this' keyword then we have to use the following syntax.

```
this.method_Name([param_List]);
```

Example programme on “this” to refer current class method:

```
class A{
void m1(){
System.out.println("m1-A");
```

```
}
void m2(){
System.out.println("m2-A");
m1();
this.m1();
}}
class Test{
public static void main(String args[]){
A a=new A();
a.m2();
}}
```

Output:

```
m2-A
m1-A
m1-A
```


3.To refer current class constructors:

- If we want to refer current class constructor by using 'this' keyword then we have to use the following syntax.

this([param_List]);

Example programme on "this" to refer current class constructors:

```
class A{
A(){
this(10);
System.out.println("A-0-arg-con");
}
A(int i){
this(22.22f);
System.out.println("A-float-int-con");
}
A(float f){
this(10.0);
System.out.println("A-float-param-con");
}
A(double d){
System.out.println("A-double-param-con");
}}
class Test{
```

```
public static void main(String args[]){
A a=new A();
}}
```

Output:

A-double-param-con

A-float-param-con

A-float-int-con

A-0-arg-con

NOTE: In the above programme we have provided more than one constructor with the same name and with the different parameter list,this process is called as "Constructor Overloading".

In the above programme,we have called all the current class constructors by using 'this' keyword in chain fashion,this proccess is called as "Constructor Chaining".

NOTE: If we want to access current class constructor by using 'this' keyword then the respective 'this' statement must be provided as first statement.

NOTE: If we want to refer current class constructor by using 'this' keyword then the respective 'this' statement must be provided in the current class another constructor,not form normal Java method.

- If we violate the above two rules then compiler will rise an error like "call to this must be first statement in constructor".

- Due to the above reasons,we are able to access only one current class constructor by using 'this' keyword from another current class constructor,it is not possible to access more than one current class constructors by using 'this' from a single current class constructor.

Example programme on,How to use “this” keyword in a valid form:

```
class A{
A(){
this(10);
this(22.22f);---->InValid
}
A(int i){
}
A(float f){
}}
```

Example programme on “this” keyword:

```
class A{
A(){
this(10);
System.out.println("A-0-arg-con");
}
A(long d){
System.out.println("A-long-param-con");
}
A(double d){
System.out.println("A-double-param-con");
}}
```

```
class Test{

public static void main(String args[]){

A a =new A();

}}
```

Output:

```
A-long-param-con

A-0-arg-con
```

- In the above application, when JVM encounter 'this(10)' then JVM will search for a constructor having matched parameter, if it is not available then JVM will search for a constructor having next higher data type parameter.
- In the above application,JVM will search for int- parameterized constructor for 'this(10)' constructor call, if it is not available then JVM will search for long-parameterized constructor, if it is not available then JVM will search for float parameterized and double parameterized constructors in the same class.

4.To return Current class Object:

Example programme on "this" to return current class Object:

```
class A{

A getRef1(){

A a=new A();

return a;

}

A getRef2(){
```

```
return this;
}}
class Test{
public static void main(String args[]){
A a=new A();//[a=abc123]
System.out.println(a);
System.out.println();
System.out.println(a.getRef1());//[abc123.getRef1()]
System.out.println(a.getRef1());//[abc123.getRef1()]
System.out.println(a.getRef1());//[abc123.getRef1()]
System.out.println();
System.out.println(a.getRef2());//[abc123.getRef2()]
System.out.println(a.getRef2());//[abc123.getRef2()]
System.out.println(a.getRef2());//[abc123.getRef2()]
}}
```

Output:

```
A@5e3a78ad

A@50c8d62f
A@3165d118
A@138297fe

A@5e3a78ad
A@5e3a78ad
A@5e3a78ad
```

- In the above programme,for every call of getRef1() method JVM will encounter "new" keyword,JVM will create new Object for class A every time and JVM will return one(new) object reference every time.This approach will increase no. of objects in Java application,it will not provide Objects reusability.
- In the above programme,for every call of getRef2() method JVM will encounter "return this" statement,JVM will not create new Object for class A every time,JVM will return the same reference value on which we calling getRef2() method.This approach will increase Objects reusability.

'static' keyword:

- Static is a Java keyword,it will **improve sharability** in Java applications.In Java applications,static keyword will be utilized in the following four ways.

  1.Static variables

  2.Static methods

  3.Static blocks

  4.Static import

1.Static variables:

- Static variables are normal Java variables,which will be recognized and executed exactly at the time of loading the respective **class bytecode** to the memory.
- Static variables are normal java variables,they will share their last modified values to the future objects and to the past objects of the respective class.
- In Java applications,static variables will be accessed either by using the respective class reference variable or by using the **respective class name directly.**

- In Java applications,static variables are always class level variables,they never be local variables.
- In Java applications,static variable vlaues will be stored in **method area**,not in Stack memory and not in Heap Memory.

NOTE:To access static variables we can use the respective class reference variable which may or may not have reference value.To access static variables it is sufficient to take a reference variable with null value.

NOTE:If we access any non-static variable by using a reference variable with null value then JVM will rise an exception like "java.lang.NullPointerException".

NOTE:In Java applications,to access current class static variables we can use "this" keyword.

Example programme on static variables:

```
class A{

static int i=10;

int j=10;

void m1(){

//static int k=30;--->error

System.out.println("m1-A");

System.out.println(this.i);

}}

class Test{

public static void main(String args[]){

A a=new A();

System.out.println(a.i);

System.out.println(A.i);

a.m1();

A a1=null;

//System.out.println(a1.j);--->NullPointerException
```

```
System.out.println(a1.i);
}}
```

Output:

```
10
10
m1-A
10
10
```

Example Programme on Static and Instance variables(allocation of variables in memory):

```
class A{
static int i=10;
int j=10;
}
class Test{
public static void main(String args[]){
A a1=new A();
System.out.println(a1.i+"  "+a1.j);
a1.i=a1.i+1;
a1.j=a1.j+1;
System.out.println(a1.i+"  "+a1.j);
```

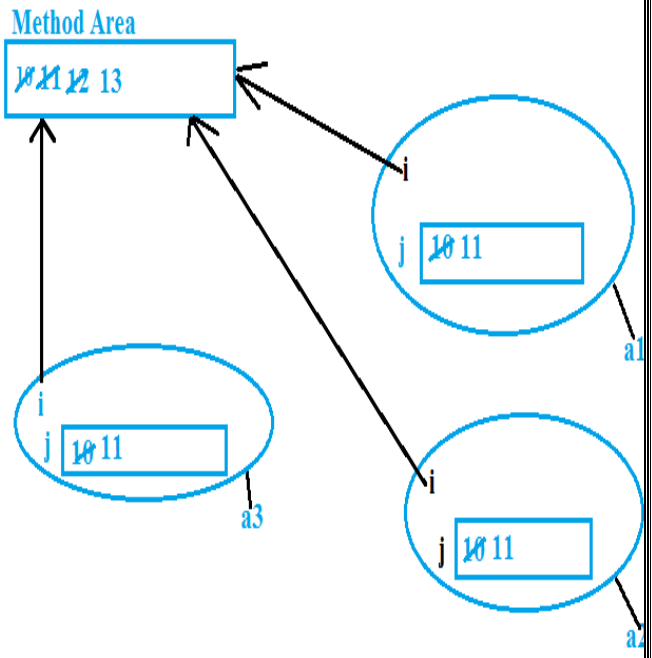
```
A a2=new A();
System.out.println(a2.i+"  "+a2.j);
a2.i=a2.i+1;
a2.j=a2.j+1;
System.out.println(a1.i+"  "+a1.j);
System.out.println(a2.i+"  "+a2.j);
A a3=new A();
System.out.println(a3.i+"  "+a3.j);
a3.i=a3.i+1;
a3.j=a3.j+1;
System.out.println(a1.i+"  "+a1.j);
System.out.println(a2.i+"  "+a2.j);
System.out.println(a3.i+"  "+a3.j);
}}
```

Output:

```
10   10
11   11
11   10
12   11
12   11
12   11
12   10
13   11
13   11
13   11
```

```
class A
{
static int i=10;
int j=10;
}
```

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

A a1=new A();                                   10      10
System.out.println(a1.i+"  "+a1.j);
a1.i=a1.i+1;
a1.j=a1.j+1;                                    11      11
System.out.println(a1.i+"  "+a1.j);
A a2=new A();                                   11      10
System.out.println(a2.i+"  "+a2.j);
a2.i=a2.i+1;
a2.j=a2.j+1;                                    12      11
System.out.println(a1.i+"  "+a1.j);
System.out.println(a2.i+"  "+a2.j);             12      11
A a3=new A();
System.out.println(a3.i+"  "+a3.j);             12      10
a3.i=a3.i+1;
a3.j=a3.j+1;
System.out.println(a1.i+"  "+a1.j);             13      11
System.out.println(a2.i+"  "+a2.j);             13      11
System.out.println(a3.i+"  "+a3.j);
                                                13      11
```



NOTE: In Java applications, Instance variable is specific to each and every object that is a separate copy of instance variables will be maintained by each and every object but static variable is common to every object that is the same copy of static variable will be maintained by all the objects of the respective class. In Java for a particular class Byte-Code will be loaded only one time but we can create any no.of objects.

2.Static Methods:

- Static method is a **normal java method**,it will be recognized and executed the time when we access that method.

- Static methods will be accessed either by using reference variable or by using the **respective class name directly**.
- In the case of accessing static methods by using reference variables,reference variable may or may not have object reference value,it is possible to access static methods with the reference **variables having 'null' value**.
- Static methods will allow only static members of the current class,static methods **will not allow non-static members** of the current class directly.
- If we want to access **non-static members** of the current class in static methods then we have to create an object for the current class and we have to use the generated reference variable.
- Static methods will not allow **'this' keyword** in its body but to access current class static methods we are able to use 'this' keyword.

Example Programme on Static methods:

```
class A{

int i=10;

static int j=20;

static void m1(){

System.out.println("m1-A");

System.out.println(j);

//System.out.println(i);---->error

//System.out.println(this.j);---------->error

A a=new A();

System.out.println(a.i);

}

void m2(){
```

```
System.out.println("m2-A");
this.m1();
}}
class Test{
public static void main(String[] args){
A a=new A();
a.m1();
a=null;
a.m1();
A.m1();
}}
```

Output:

```
m1-A
20
10
m1-A
20
10
m1-A
20
10
```

Q)Is it possible to print a line of text on command prompt with out using main() method?

Ans:Yes,it is possible to display a line of text on command prompt with out using a static variable and static method.

Example programme on without main() method:

```
class Test{

static int i=m1();

static int m1(){

System.out.println("Welcome to durga software soultions");

System.exit(0);//to terminate the application

return 10;

}}
```

Output:

Welcome to durga software soultions

- If we provide 'Test' class name along with 'java' command on command prompt then JVM will take main class from command prompt,JVM will search for its **.class file**,if it is available then JVM will load main class bytecode to the memory that is Test class bytecode.At the time of loading Test class bytecode to the memory,JVM will recognize and initialize static variable,as part of initialization,JVM will execute static method.
- By the execution of static method,JVM will display the required message on command prompt, when JVM encounter **System.exit(0)** statement then JVM will terminate the application.

NOTE:The above question and answer is valid upto JAVA6 version,it is not valid in JAVA7 version,because,In JAVA6 version JVM will load main class bytecode

to the memory irrespective of main() method availability.In JAVA7,JVM will load main class bytecode to the memory with main() method availability only.If we run the above code in JAVA7 version then JVM will display the following error message.

Error:Main Method not found in class Test,please define the main method as:

public static void main(String args[])

3.Static Block:

- Static Block is a set of instructions,which will be recognized and executed at the time of loading the respective class bytecode to the memory.
- Static blocks will allow static members of the current class directly.
- Static blocks will not allow non-static members of the current class directly,where if we want to access non-static members in static block then we must create object for the current class and we have to use the generated reference variable.
- Static blocks will not allow 'this' keyword in its body.

Example programme on Static Blocks:

```
class A{

int i=10;

static int j=20;

static{

System.out.println("SB-A");

System.out.println(i);//-------->Error
```

```
A a=new A();

System.out.println(a.i);

System.out.println(j);

System.out.println(this.j);-------->Error

}}
```

Output:

```
SB-A

10

20

Exception in thread "main" java.lang.NoSuchMethodError:main
```

Q)Is it Possible to print a line of text on command prompt with out using main() method,static variable and static method?

Ans:Yes,it is possible to display a line of text on command prompt without using main() method,static variable,static method but by using static block.

Example Programme on with out using main() method,static variable and static method:

```
class Test{

static{

System.out.println("Welcome to DurgaSoftware Soultions");

System.exit(0);//To terminate the programme

}}
```

Output:

Welcome to DurgaSoftware Soultions

- If we provide main class name 'Test' along with 'java' command on command prompt then JVM will take main class name i.e Test and JVM will search for its .class file.If Test.class file is identified then JVM will load its bytecode to the memory,at the time of loading bytecode to the memory static block will be executed,with this,the required message will be displayed on command prompt.When JVM encounter System.exit(0) then JVM will terminate the programme.

NOTE:The above question and answer is valid upto JAVA6 version,it is invalid in JAVA7 version, because,in JAVA6 version JVM will load main class bytecode to the memory with out checking main() method availability but in JAVA7 JVM will load main class bytecode to the memory after checking main() method availability.If main() method is available then only JVM will load main class bytecode otherwise JVM will not load main class bytecode.

If we use JAVA7 version to run the above programme then JVM will provide the following message on command prompt.

Error:Main method not found in class Test,please define main method as:public static void main(String args[])

Q)Is it possible to display line of text on command prompt with out using main() method,static variable,static method,static block?

Ans:Yes,it is possible to display a line of text on command prompt with out using main() method,a static variable,a static method and static block but by using "static Anonymous Inner class".

Example programme with out using main() method,static variable,static method,static block:

class Test{

```
static Object obj=new Object(){

{

System.out.println("Welcome To durga Software Solutions");

System.exit(0);//TO termiante the application.

}};}
```

Output:

JAVA6:Welcome to Durga Software Soultions

JAVA7:Error:main method not found in class Test,please define the main method as:public static void main(String[] args)

4.static import:

- In Java,applications,if we import a particular package to the present java file then it is possible to access all the classes and interfaces of that package directly with out using package name every time as fully qualified name.
- If we want to access classes and interfaces of a particular package with out importing then we must use fully Qualified name every time that we have to use package name along with class names.

With out import statement:

```
java.io.BufferedReader br=new java.io.BufferedReader(new
java.io.InputStreamReader(System.in));
```

With import statement:

```
import java.io.*;

BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
```

- In Java applications,if we want to access static members of a particular class in present java file then we must use **either reference variable of the respective class or directly class name**.
- In Java applications,if we want to access static members without using the respective class name and with out using the respective class reference variable then we have to import static members of that respective class in the present Java file.
- To import static members of a particular class in the present java file,**JDK 5.0 version** has provided a new feature called as "static import".

Syntax:

import static package_Name.Class_Name.*;

It will import all the static members from the specified class.

Ex:import static java.lang.Thread.*;

import static package_Name.Class_Name.member_Name;

It will import only the specified member from the specified class.

Example programme on "static import":

```
import static java.lang.Thread.MIN_PRIORITY;
import static java.lang.Thread.*;
import static java.lang.System.out;
    class Test{
    public static void main(String args[]){
    out.println(MIN_PRIORITY);
    out.println(MAX_PRIORITY);
    out.println(NORM_PRIORITY);
```

```
}}
```

Output:

```
1
10
5
```


Static Context / Static Flow of Execution:

In Java,Static Context will be represented in the form of the following three elements.

1.Static variables.

2.Static methods.

3.Static blocks

- In Java applications,instance context will be created separately for each and every object but **static context will be created for each and every class.**
- In Java applications,static context will be recognized and executed exactly **at the time of loading the respective class bytecode** to the memory.
- In Java applications,when we create object for a particular class,first,JVM has to access constructor,before executing constructor,JVM has to load the respective class bytecode to the memory.At the time of loading class bytecode to the memory,**JVM has to recognize and execute static context.**

Example programmes on static context or static Flow of Execution:

```
class A{
```

```
static {
System.out.println("SB-A");
}
static int i=m1();
static int m1(){
System.out.println("m1-A");
return 10;
}}
class Test{
public static void main(String args[]){
A a=new A();
}}
```

Output:

```
SB-A
m1-A
```

```
class A{
static int i=m2();
static int m1(){
System.out.println("m1-A");
return 10;
}
static{
System.out.println("SB-A");
}
```

```
static int m2(){
System.out.println("m2-A");
return 20;
}
static int j=m1();
}

class Test{
public static void main(String args[]){
A a1=new A();
A a2=new A();
}}
```

Output:

```
m2-A
SB-A
m1-A
```

```
class A{
static int i=m2();
A(){
System.out.println("A-con");
}
int m1(){
System.out.println("m1-A");
return 10;
}
```

```
static{
System.out.println("SB-A");
}
int j=m1();{
System.out.println("IB-A");
}

static int m2(){
System.out.println("m2-A");
return 10;
}}
class Test{
public static void main(String args[]){
A a1=new A();
A a2=new A();
}}
```

Output:

```
m2-A
SB-A
m1-A
IB-A
A-con
m1-A
IB-A
A-con
```

```
class A{
private A(){
System.out.println("A--con");
}
void m1(){
System.out.println("m1--A");
}}
class Test{
public static void main(String args[]){
A a=new A();
}}
error: Test.java:10: A() has private access in A
        A a=new A();
```


Factory Method:

- Factory Method is a method,it can be used to **return the same class Object** where we have declared that method.
- Factory Method is an idea provided by a design pattern called as "**Factory Method Design Pattern**".

NOTE:Design pattern is a System,it will provide problem definition and its solution inorder to resolve design problems.

Example Programme On Factory Method:

```
class A{
private A(){
System.out.println("A-con");
}
void m1(){
System.out.println("m1-A");
}
static A getRef()//Factory Method
{
A a=new A();
return a;
}}
class Test{
public static void main(String args[]){
A a=A.getRef();
a.m1();
}}
```

Output:

```
A-con
m1-A
```

There are two types of Factory Methods

1.Static Factory Method

2.Instance Factory Method

Static Factory Method:

- Static Factory Method is a static method returns the same class object.

Ex:

```
NumberFormat nf=NumberFormat.getInstance(-);

DateFormat df=DateForamt.getInstance(--);

ResourceBundle rb=ResourceBundle.getBundle(--);
```

Instance Factory Method:

- If any non-static method returns the same class object then that method is called as "Instance Factory Method".

  Ex:Allmost all the String class methods are Instance Factory methods.

```
String str=new String("DurgaSoftware Solutions");

String str1=str.concat(" Hyderabad");

String str2=str.trim();

String str3=str.toUpperCase();

String str4=str.substring(5,14);
```

Singleton Class:

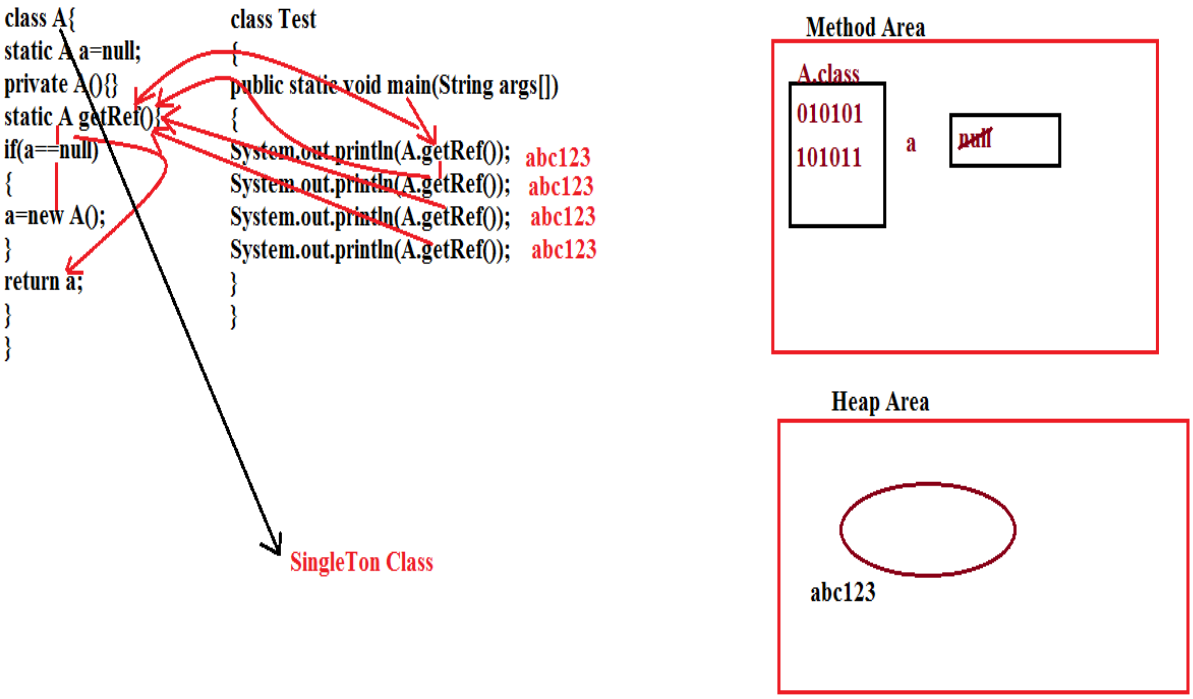
- If any JAVA class allows to **create only one Object** then that class is called as "Singleton Class".
- Singleton Class is an idea provided by a design pattern called as "**Singleton Design Pattern**".

Example programme on Singleton class:

```
class A{

static A a=null;

private A(){

}

static A getRef(){

if(a==null){

a=new A();

}

return a;

}}

class Test{

public static void main(String args[]){

System.out.println(A.getRef());

System.out.println(A.getRef());

System.out.println(A.getRef());

}}
```

Output:

```
A@a6eb38a

A@a6eb38a

A@a6eb38a
```



Example programme on,Alternative Logic for Singleton Class:

```
class A{

static A a=new A();

private A(){

}

static A getRef(){
```

```
return a;
}}
class Test{
public static void main(String args[]){
System.out.println(A.getRef());
System.out.println(A.getRef());
System.out.println(A.getRef());
}}
```

Output:

```
A@69cd2e5f
A@69cd2e5f
A@69cd2e5f
```


Example programme on,Another Alternative for Singleton Class:

```
class A{
static A a=null;
static{
a=new A();
```

```
}
private A(){
}
static A getRef(){
return a;
}}
class Test{
public static void main(String args[]){
System.out.println(A.getRef());
System.out.println(A.getRef());
System.out.println(A.getRef());
}}
```

Output:

```
A@a6eb38a
A@a6eb38a
A@a6eb38a
```


final Keyword:

final is a Java Keyword it can be used to declare constant expressions.

In Java applications,there are three ways to utilize 'final' keyword.

1.final variable

2.final method

3.final class

1.final variable:

final variable is a variable,it will not allow modifications on its value.

Ex:

```
final int i=10;
i=i+10;
```

Output:

CE: Test.java:4: cannot assign a value to final variable i

i=i+10;

1 error

Ex:

```
for(final int i=0;i<10;i++)->error
{
System.out.println(i);
}
```

Output:

CE: **Test.java:2: illegal start of type**

for(final int i=0;i<10;i++)

^

Test.java:2: ')' expected

for(final int i=0;i<10;i++)

NOTE:In general, in bank applications, after creating an account it is possible to change the account details like account name, address details....but it is not possible to update 'accNo' value once it is created. Due to this reason, we have to declare 'accNo' variable as 'final'.

2.final method:

final method is a Java method,it will not allow method overriding.

3.final class:

final class is a Java class,it will not allow inheritance.In Java applications,super classes never be declared as final classes,but subclasses may be final.

final class A

{

```
}
class B extends A
{
}
```

Status: Invalid

```
class A
{
}
```

```
final class B extends A
{
}
```

Status: Valid

In Java applications to declare constant variables Java has provided a convention like to declare constants with "public static final".

\Examples programmes on “final” keyword:

System:

```
public static final PrintStream out;
public static final InputStream in;
```

public static final PrintStream err;

Thread:

public static final int MIN_PRIORITY=1;

public static final int NORM_PRIORITY=5;

public static final int MAX_PRIORITY=10;

Example Programme on to declare constants with "public static final":

```
class MailStatus{
public static final String AVAILABLE="Available";
public static final String BUSY="Busy";
public static final String IDLE="Idle";
}
class Test{
public static void main(String args[]){
System.out.println(MailStatus.AVAILABLE);
System.out.println(MailStatus.BUSY);
System.out.println(MailStatus.IDLE);
}}
```

Output:

Available

Busy

Idle

To declare constant variables in Java applications if we use the above convention then we are able to get the following problems.

1.We must declare "public static final" for each and every constant variable explicitly.

2.It is possible to allow multiple data types to represent one type,it will reduce typedness feature for Java applications.

3.If we access constant variables then these variables will display their values,here constant variable values may or may not reflect the actual meaning of constant variables.

To overcome all the problems,we have to go for "enum".

In case of enum,

1.All the constant variables are by default "public static final",no need to declare "public static final" explicitly.

2.All the constant variables are by default the same enum type,it will improve Typedness in Java applications.

3.All the constant variables are by default "Named Constants" that is,these constant variables are displaying their names instead of their values.

Example programme on "enum" keyword:

```
enum MailStatus{

AVAILABLE,BUSY,IDLE;
```

```
}
class Test{
public static void main(String args[]){
System.out.println(MailStatus.AVAILABLE);
System.out.println(MailStatus.BUSY);
System.out.println(MailStatus.IDLE);
}}
```

Output:

Available

Busy

Idle

NOTE:The default super class for every enum is "java.lang.Enum" class and "Object" class is Super class to "Enum" class.

Internal Flow:

If we compile the above Java file then Compiler will translate "MailStatus" enum into "MailStatus" final class like below.

```
final class MailStatus extends java.lang.Enum{
public static final MailStatus AVAILABLE;
public static final MailStatus BUSY;
```

public static final MailStatus IDLE;

}

NOTE: In Java,java.lang.Object class is common and default super class for all the classes.Similarly,All the Java enums are having a common and default super class that is "java.lang.Enum".

In Java applications,it is possible to implement inheritance between two classes but it is not possible to implement inheritance between two "enums".

In Java applications,we can utilize enum like as classes,where we can provide normal variables,methods,constructors....

Example programme on "enum" keyword by using reference :

enum Apple{

A(500),B(250),C(100);

int price;

```
Apple(int price){
this.price=price;
}
public int getPrice(){
return price;
}}
```


```
class Test{
public static void main(String args[]){
System.out.println("A-Grade Apple  :"+Apple.A.getPrice());
System.out.println("B-Grade Apple  :"+Apple.B.getPrice());
System.out.println("C-Grade Apple  :"+Apple.C.getPrice());
}}
```

Output:

```
A-Grade Apple  :500
B-Grade Apple  :250
C-Grade Apple  :100
```

If we compile the above programme,then compiler will translate enum into the following class:

Translated Code for the above enum(Apple):

```
final class Apple extends Enum{
public static final Apple A=new Apple(500);
```

```
public static final Apple B=new Apple(250);
public static final Apple C=new Apple(100);
int price;
Apple(int price){
this.price=price;
}
public int getPrice(){
return price;
}
----
}
```

Another Example programme on "enum" keyword by using reference :

```
enum Book{
A(500,250),B(300,150),C(200,100);
int no_of_pages;
int cost;
Book(int no_of_pages,int cost){
this.no_of_pages=no_of_pages;
```

```
this.cost=cost;
}
public void getBookDetails(){
System.out.println(no_of_pages+"--------->"+cost);
}}
class Test{
public static void main(String args[]){
System.out.println("Durga Books Store");
System.out.println("-------------------");
System.out.println("No of Pages   Cost");
System.out.println("-------------------");
Book.A.getBookDetails();
Book.B.getBookDetails();
Book.C.getBookDetails();
}}
```

Output:

```
Durga Books Store
-------------------------
No of Pages Cost
-------------------------
500------------>250
300------------>150
200------------>100
```

If we compile the above program,then compiler will translate the enum Into the following class

Translated Code for the above enum(Book):

```
final class Book extends Enum{
public static final Book A=new Book(500,250);
public static final Book B=new Book(300,150);
public static final Book C=new Book(200,100);
int cost;
int no_of_pages;
Book(int no_of_pages,int cost){
this.no_of_pages=no_of_pages;
this.cost=cost;
}
public void getBookDetails(){
System.out.println(no_of_pages+"--------->"+cost);
}
----
}
```


Class.forName(--):

Consider the following programme:

```
class A{
static{
System.out.println("Class Loading");
```

```
}

A(){

System.out.println("Object Creating");

}}

class Test{

public static void main(String args[]){

A a=new A();

}}
```

In the above programme,when we access 'A' class constructor along with 'new' keyword then JVM will perform automatically the following two actions.

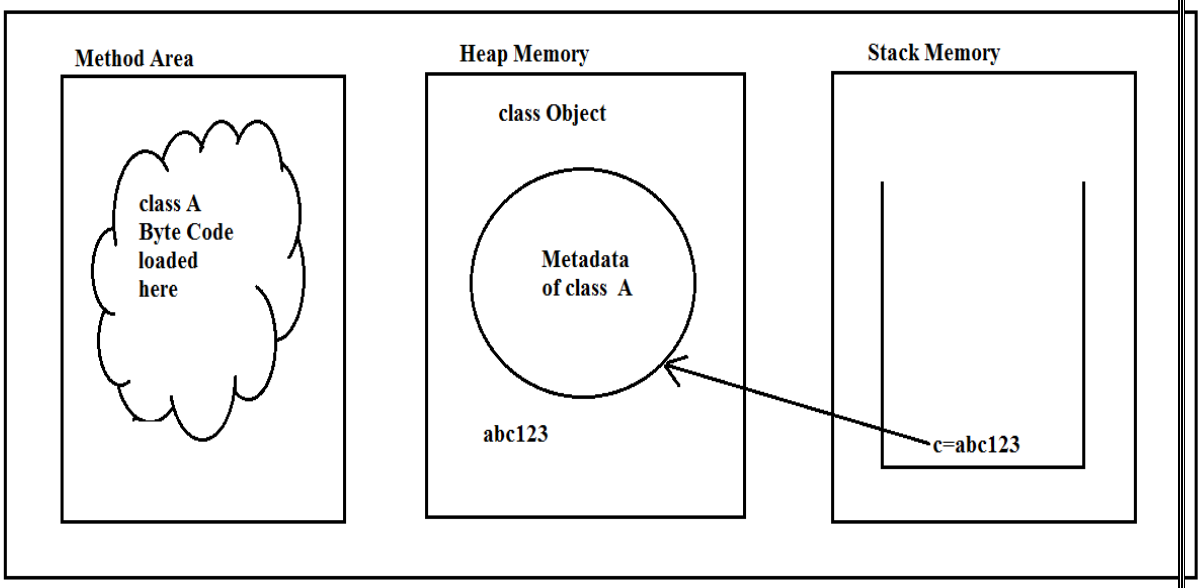
1.Loading the class bytecode to the memory.

2.Creating object for the respective class

In the above context,as per the application requirement we want to load the respective class bytecode to the memory without creating object.To achieve this requirement,we have to use the following method from "java.lang.Class" class.

public static Class forName(String class_Name) throws ClassNotFoundException

Ex:Class c=Class.forName("A");



When JVM encounter the above instruction,JVM will perform the following actions.

1.JVM will take the provided class name from "**forName()**" method.

2.JVM will search for the respective **.class file** at current location,at Java predefined library and at the locations refereed by "classpath" environment variable.

3.If the required .class file is not available at the above three locations then JVM rise an exception like "java.lang.ClassNotFoundException".

4.If the required **.class file** is available at either of the above three locations then JVM will load its bytecode to the memory and JVM will collect the loaded class metadata and JVM will manage that metadata in the form **"java.lang.Class"** object at heap memory.

After loading class bytecode to the memory by using "forName(-)" method,if we want to create object for the loaded class explicitly then we have to use the following method from "java.lang.Class" class.

public Object newInstance() throws InstantiationException,IllegalAccessException

Ex:Object obj=c.newInstance();

When JVM encounter the above instruction,JVM will perform the following actions.

1.JVM will go to the respective "Class" object in Heap memory.

2.JVM will search for non-private and 0-arg constructor in the "Class" object.

3.If the required constructor is available then JVM will execute it and JVM will create object for the respective class in Heap Memory.

4.If the constructor is not 0-argument,if the constructor is parametrized constructor then JVM will rise an exception like "java.lang.InstantiationException".

5.If the constructor is private then JVM will rise an exception like

"java.lang.IllegalAccesException".

6.If the constructor is private and parametrized constructor the JVM will rise only one exception that is "java.lang.InstantiationExecption".

Example programme on creating Object using newInstance():

```
class A{

static{

System.out.println("Class Loading");

}

A(){

System.out.println("Object creating");

}}

class Test{

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

```
Class c=Class.forName("A");

Object obj=c.newInstance();

}}
```

Output:

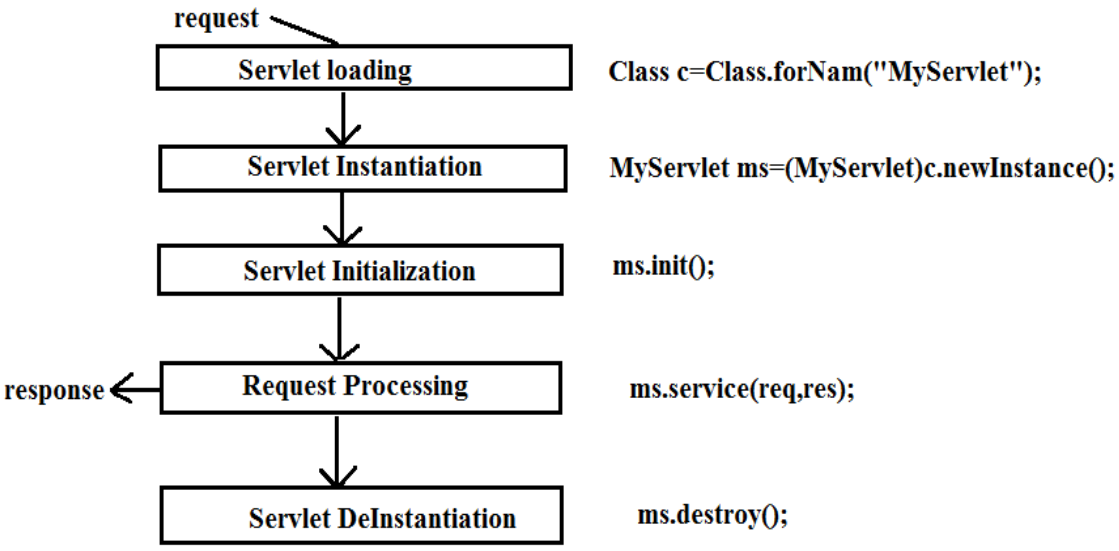
Class Object

Object creating

NOTE:In Jdbc Applications,to perform database operations it is mandatory to load driver class bytecode to the memory.To load Driver class bytecode to the memory we have to use Class.forName() method.

```
Ex:Class.forName("oracle.jdbc.OracleDriver");

   Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
```

NOTE:"Servlet" is a server side componet in web applications,it will be executed by the Container[A part in Server] by following lifeCycle actions with out using main() method.To perform Servlet life cycle actions like "Servlet loading","Servlet Instantiation" container will execute Class.forName() and newInstance() internally.



Importance of main() method in Java:

The main intention of main() method in Java applications is,

1.To define application logic in Java programme.

2.To define starting point and ending point for the applications execution.

Syntax:

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

```
----application logic---

}
```

- main() method is not predefined method and user defined method,it is a conventional method with fixed prototype and with user defined implementation.
- In Java,JVM is implemented in such a way to access main() method automatically in order to execute application logic.

Q)What is the requirement to declare main() method as public?

- In Java applications,JVM has to access main() method inorder to start application execution.
- To access main() method by JVM first main() scope must be available to JVM.In this context,to bring main() method scope to JVM,we must declare main() method as "public".

Case-1:

- If main() method is declared as "private" then main() method will be available upto the main class,not to JVM.

Case-2:

- If main() method is declared as "default" then main() method will be available upto the package where main class is existed,not to the JVM.

Case-3:

- If main() method is declared as "protected" then main() method will be available upto the package where main class is existed and upto child classes available other packages but not to the JVM.

Case-4:

- To make available main() method JVM,only one possibility we have to take that is "public",public members will be available through out our system,so that,JVM can access main() method to start application execution.

NOTE:In Java applications,if we declare main() method without "public" then compiler will not rise any error but JVM will provide the following.

JAVA6:Main Method not public.

JAVA7:Error:Main method not found in class Test,please define main method as:public static void main(String args[])

Q)What is the requirement to declare main() method as "static"?

- Ans:In Java applications,to start application execution JVM has to access main() method.JVM was implemented in such a way that to access main() method by using the respective main class directly.
- In Java applications,only static methods are eligible to access by using their respective class name,so that,as per the JVM predefined implementation we must declare main() method as static.

NOTE:In Java applications,if we declare main() method without "static" then compiler will not rise any error but JVM will provide the following.

JAVA6:java.lang.NoSuchMethodError:main

JAVA7:Error:Main method is not static in class Test,please define main method as:public static void main(String[] args)

Q)Is it possible to interchange "public" and "static" in main() method syntax?

- **Ans**:Yes,it is possible to interchange "public" and "static" in main() method syntax,because, normal Java methods will allow more than one access modifier in any order but in valid combination.

  public static void main(String args[]) -->valid

  static public void main(String args[]) -->Valid

  public static final main(String[] args) -->Valid

public static abstract void main(String args[]) -->invalid

public void static main(String[] args) -->Invalid

Q)What is the requirement to provide "void" as return type to main() method?

- **Ans**:In Java applications,as per Java conventions,JVM will start application execution at the starting point of main method and JVM will terminate application execution at the ending point of main() method.Due to this convention,we must start application logic at the starting point of main() method and we must terminate application logic at the ending point of main() method.To follow this Java convention we must not return any value from main() method for this,we must provide "void" as return type.

NOTE:In Java applications,if we declare main() method with out void return type then compiler will not rise any error but JVM will provide the following

JAVA6: java.lang.NoSuchMethodError:main

JAVA7:Error:Main method must return a value of type void in class Test,please define the main method as:

public static void main(String[] args)

NOTE:The name of this method "main" is to show the importance of this method.

Main method parameters:

Q)What is the requirement to provide parameter to main() method?

- Ans:In Java applications,there are three ways to provide input to the Java programs.

  1.Static Input

2.Dynamic Input

3.Command Line Input

1.Static Input:

- Providing input to the Java program at the time of writing Java program.

Ex:

```
int i=10;
int j=20;
void add()
{
System.out.println(i+j);
}
```


2.Dynamic Input:

- Providing Input to the Java program at runtime or at execution time.

D:\Java7>javac Add.java

D:\java7>java Add

First value : 10

Second value :20

Addition :30

3.Command Line Input:

- Providing input to the Java program along with "java" command on command prompt.

Ex:

D:\java7>javac Add.java

D:\java7>java Add 10 20

Addition : 30

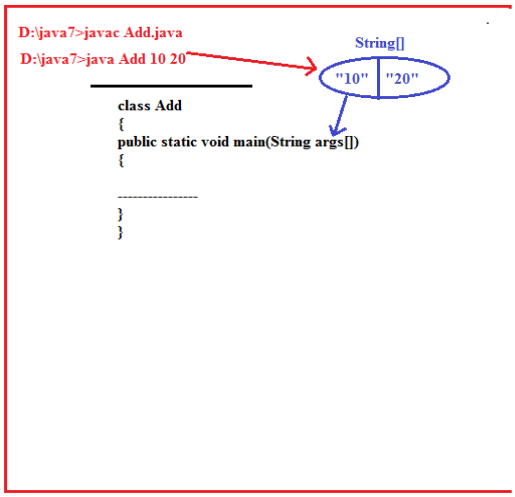
If we provide command line input like above in Java applications then JVM will perform the following actions.

a)JVM will read all the command line input at the time of reading main class name from command prompt.

b)JVM will store all the command line inputs in the form of String[]

c)JVM will pass the generated String[] as parameter to main() method at the time of calling main() method.

Due to the above JVM actions,the main requirement to provide main() method parameters is to store all the command line inputs in the form String[] array.



Q)What is the requirement to provide only String data type as parameter to the main() method?

- **Ans**:In general,from application to application or from developer to developer the types of command line input may be varies,here even developers are providing different types of command line input our main() method must store all the command line input.In Java only String data types is able to represent any type of data so that String data types is required as parameter to main() method.
- To allow different types of command line input main() method must require String dataType.

Q)What is the requirement to provide an array as parameter to main() method?

- **Ans**:In general,from application to application and from developer to developer no.of command line inputs may be varied,here even developers are providing variable no.of command line inputs our main() method parameter must store them.In Java,to store more than one value we have to take array.Due to this reason,main() method must require array type as parameter.
- main() method will take array type as parameter is to follow multiple no.of command line input.

Example programmes on Command Line Input():

```
class Test{
public static void main(String args[]){
for(int i=0;i<args.length;i++){
System.out.println(args[i]);
}}
```

O/P:D:\java7>javac Test.java

```
   D:\java7>java Test 10 "abc" 22.22f 34.345 'A' true
        10
        "abc"
        22.22f
        34.345
        'A'
        true
class Test{
```


```
public static void main(String args[])throws Exception{
int fval=Integer.parseInt(args[0]);
int sval=Integer.parseInt(args[1]);
System.out.println("Addition :"+(fval+sval));
```

```
System.out.println("Subtraction :"+(fval-sval));

System.out.println("Multiplication :"+(fval*sval));

}}
```

O/P:D:\java7>javac Test.java

D:\java7>java Test 10 5

Addition :15

Subtraction :5

Multiplication:50

- If we declare main() method without String[] parameter then compiler will not rise any error but JVM will provide the following.

JAVA6:java.lang.NoSuchMethodError:main

JAVA7:Error:Main Method not found in class Test,please define main method as:public static void main(String args[])

Q)Find the valid syntaxes of main() method from the following list?

1.public static void main(String[] args) --->valid

2.public static void main(String[] abc) --->valid

3.public static void main(String args[]) --->valid

4.public static void main(String []args) --->valid

5.public static void main(string[] args) --->Invalid

6.public static void Main(String[] args) --->Invalid

7.public static int main(String[] args) --->Invalid

8.public static final void main(String[] args) --->valid

9.public final void main(String[] args) --->Invalid

10.static public void main(String[] args) --->valid

11.static void main(String[] args) --->Invalid

12.public static void main(String ... args) --->valid

Q)Is it possible to provide more than one main() method with in a single java application?

- **Ans**:Yes,is it possible to provide more than one main() method with in a single java application,but,we have to provide more than one main() method in different classes,not in a single class.

  Ex:

  D:\java7\abc.java

```
class A{
public static void main(String args[]){
System.out.println("main()-A");
}}
class B{
public static void main(String args[]){
System.out.println("main()-B");
}}
```

  O/P:D:\java7>javac abc.java

  D:\java7>java A

  main()-A

  D:\java7>java B

main()-B

NOTE: If we compile the above abc.java file then compiler will generate two .class files[A.class,B.class].To execute the above program,we have to give a class name along with "java" command,here which class name we are providing along with "java" command that class main() method will be executed by JVM.

NOTE: In the above program,it is possible to access one main() method from another main() method by passing String[] as parameter and by using the respective class name as main() method is static method.

Ex:

D:\java7\abc.java

```
class A{
public static void main(String args[]){
System.out.println("main()-A");
String[] str={"AAA","BBB","CCC"};
B.main(str);
B.main(args);
}}
class B{
public static void main(String args[]){
System.out.println("main()-B");
}}
```

O/P:D:\java7>javac abc.java

D:\java7>java A

main()-A

main()-B

main()-B

Q)Is it possible to overload main() method?

- **Ans**:In Java,it is possible to overload main() method but it is not possible to override main() method,because,in Java applications static method overloading is possible but static method overriding is possible but static method overriding is not possible.

  Example programme on overload of main() method:

  ```
  class Test{
  public static void main(String args[]){
  System.out.println("String[]-param-A");
  }
  public static void main(int[] args){
  System.out.println("int[]-param-A");
  }}
  ```

  Output:

  String[]-param-A

Relationships in JAVA:

As part of Java application development,we have to use entities as per the application requirements.

In Java application development,if we want to provide optimizations over memory utilization,code reusability,execution time,sharability then we have to define relationships between entities.

There are three types of relationships between entities.

1.Has-A relationship

2.IS-A relationship

3.USE-A relationship

1.Has-A relationship:

Has-A relationship will define associations between entities in Java applications,here associations between entities will improve communication between entities and it will improve data navigation between entities.

There are four types of associations between entities

1.One-To-One Association

2.One-To-Many Association

3.Many-To-One Association

3.Many-To-Many Association

To achieve associations between entities,we have to declare either single reference or array of references of an entity class in another entity class.

Ex:

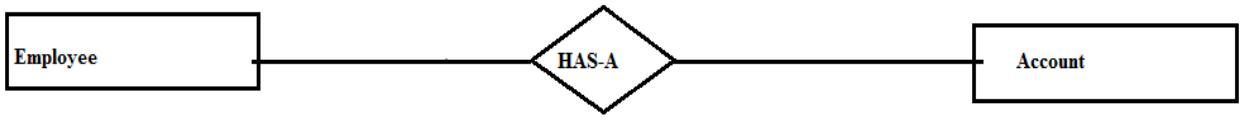
```
class Address{

----

}

class Student{

----

Address[] addr;-->It will establish One-To-Many Association

}
```


1.One-To-One Association:

It is a relationship between entities,where one instance of an entity should be mapped with exactly one instance of another entity.

Ex:

Every employee should have exactly one Account.



Example programme on One-To-One Association:

```
class Account{
String accNo;
String accName;
String accType;
Account(String accNo,String accName,String accType){
this.accNo=accNo;
this.accName=accName;
this.accType=accType;
}}
class Employee{
String eid;
String ename;
String eaddr;
Account acc;
Employee(String eid,String ename,String eaddr,Account acc){
this.eid=eid;
this.ename=ename;
this.eaddr=eaddr;
this.acc=acc;
```

```
}
public void getEmployee(){
System.out.println("Employee Details");
System.out.println("-----------------");
System.out.println("Employee Id      :"+eid);
System.out.println("Employee Name    :"+ename);
System.out.println("Employee Address :"+eaddr);
System.out.println();
System.out.println("Account Details");
System.out.println("----------------");
System.out.println("Account Number    :"+acc.accNo);
System.out.println("Account Name      :"+acc.accName);
System.out.println("Account Type      :"+acc.accType);
}}
class OneToOneEx{
public static void main(String args[]){
Account acc=new Account("abc123","Durga N","Savings");
Employee emp=new Employee("E-111","Durga","Hyd",acc);
emp.getEmployee();
}}
```

Output:

```
Employee Details
-----------------------
Employee Id  :E-111
Employee Name:Durga
```

Employee Address:Hyd

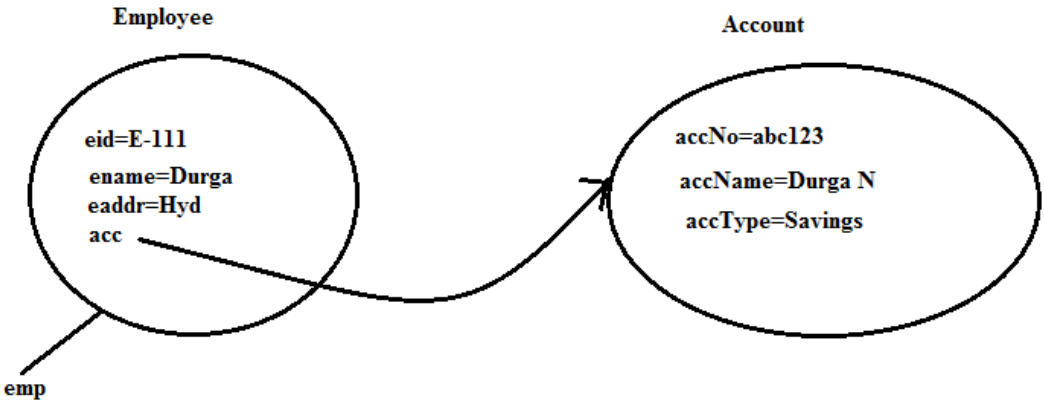
Account Details

Account Number :abc123

Account Name :Durga N

Account Type :Savings

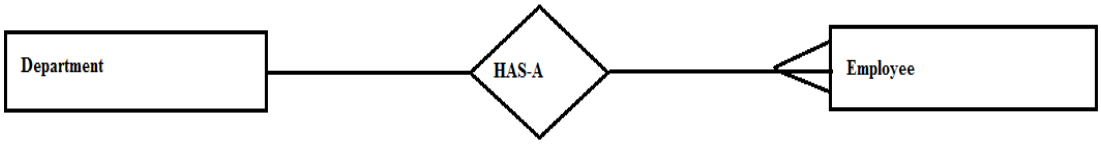
Data Flow Diagram on One-To-One:



2.One-To-Many Association:

It is a relationship between entity classes,where one instance of an entity should be mapped with multiple instances of another entity.

Ex:Single department has multiple employees.



Example Programme on One-To-Many:

```
class Employee{

String eid;

String ename;

String eaddr;
```

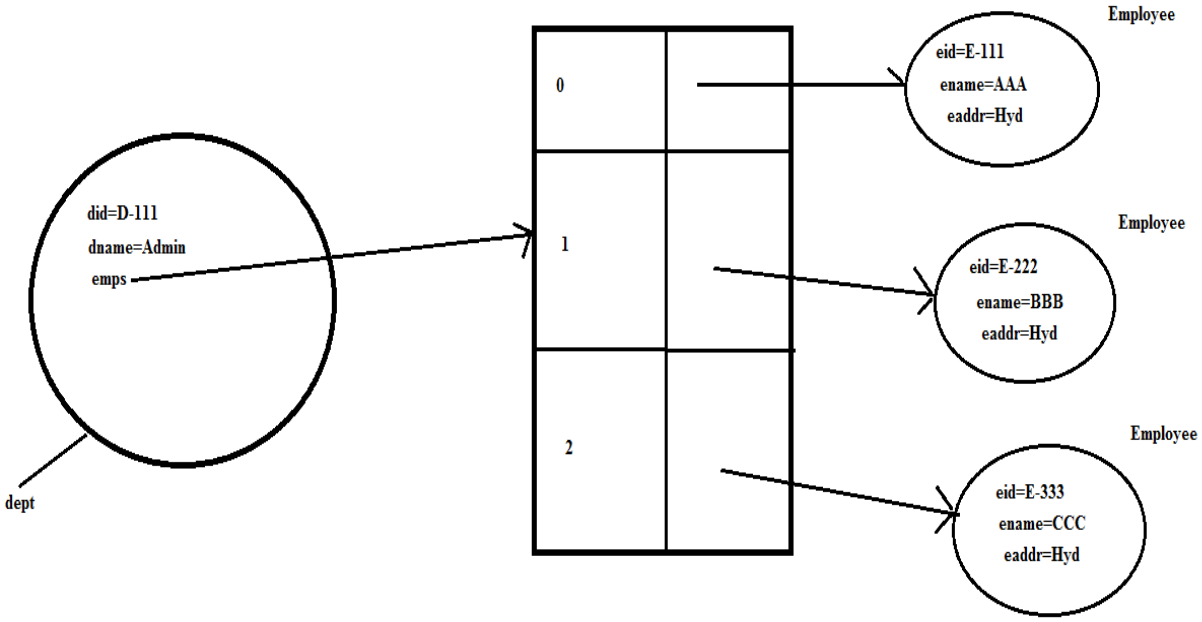
```
Employee(String eid,String ename,String eaddr){
this.eid=eid;
this.ename=ename;
this.eaddr=eaddr;
}}
class Department{
String did;
String dname;
Employee[] emps;
Department(String did,String dname,Employee[] emps){
this.did=did;
this.dname=dname;
this.emps=emps;
}
public void getDepartmentDetails(){
System.out.println("Department Details");
System.out.println("--------------------");
System.out.println("Department Id :"+did);
System.out.println("Department Name:"+dname);
System.out.println();
System.out.println("EID ENAME EADDR");
System.out.println("----------------");
for(int i=0;i<emps.length;i++){
Employee e=emps[i];
System.out.println(e.eid+" "+e.ename+" "+e.eaddr);
}}}
```

```
class OneToManyEx{
public static void main(String args[]){
Employee e1=new Employee("E-111","AAA","Hyd");
Employee e2=new Employee("E-222","BBB","Hyd");
Employee e3=new Employee("E-333","CCC","Hyd");
Employee[] emps=new Employee[3];
emps[0]=e1;
emps[1]=e2;
emps[2]=e3;
Department dept=new Department("D-111","Admin",emps);
dept.getDepartmentDetails();
}}
```

Output:

```
Department Details
---------------------------
Department Id :D-111
Department Name:Admin

EID    ENAME  EADDR
-------------------------------
E-111    AAA       Hyd
E-222    BBB       Hyd
E-333    CCC       Hyd
```

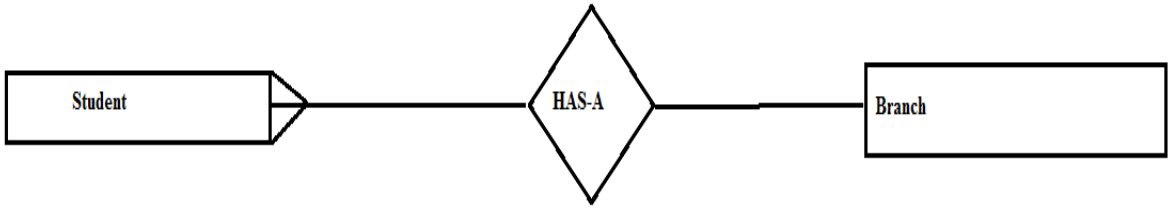
Data Flow Diagram on One-To-Many:



Many-To-One Association:

It is a relationship between entities,where multiple instances of an entity should be mapped with exactly one instance of another entity.

Ex:Multiple Student have joined with a single branch.



Example programme on Many-To-One:

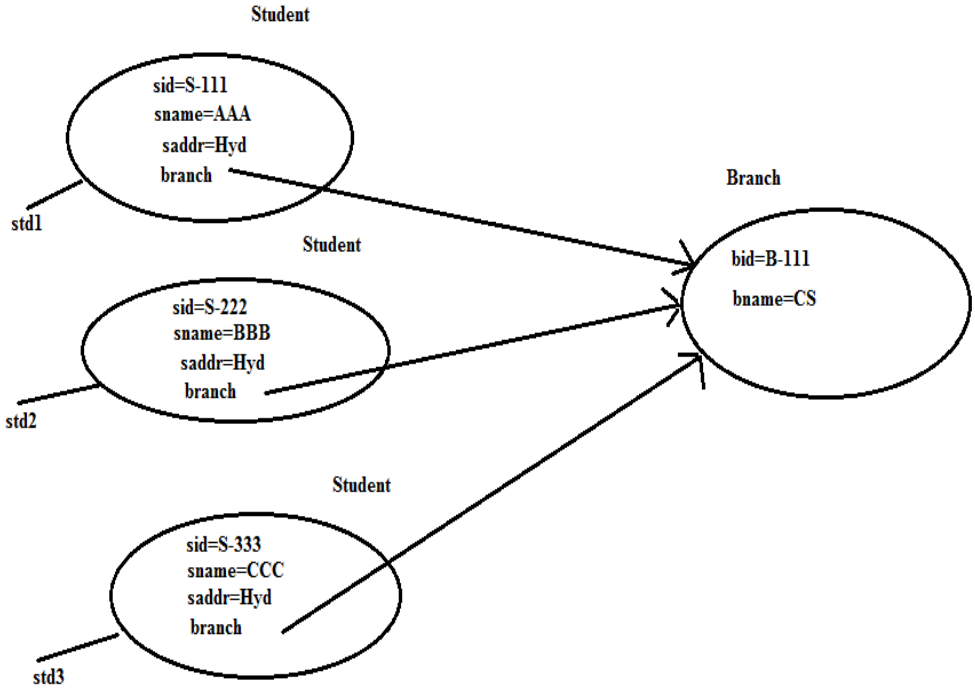
```
class Branch{
String bid;
String bname;
Branch(String bid,String bname){
this.bid=bid;
this.bname=bname;
}}
class Student{
String sid;
String sname;
String saddr;
Branch branch;
Student(String sid,String sname,String saddr,Branch branch){
this.sid=sid;
this.sname=sname;
this.saddr=saddr;
this.branch=branch;
}
public void getStudentDetails(){
```

```
System.out.println("Student Details");
System.out.println("----------------");
System.out.println("Student Id    :"+sid);
System.out.println("Student name  :"+sname);
System.out.println("Student Address:"+saddr);
System.out.println("Branch Id    :"+branch.bid);
System.out.println("Branch Name :"+branch.bname);
System.out.println();
}}
class ManyToOneEx{
public static void main(String args[]){
Branch branch=new Branch("B-111","CS");
Student std1=new Student("S-111","AAA","Hyd",branch);
Student std2=new Student("S-222","BBB","Hyd",branch);
Student std3=new Student("S-333","CCC","Hyd",branch);
std1.getStudentDetails();
std2.getStudentDetails();
std3.getStudentDetails();
}}
```

Output:

```
Student Details
----------------------
Student Id :S-111
Student name:AAA
Student Address:Hyd
Branch Id  :B-111
Branch Name:CS
Student Details
------------------------
Student Id  :S-222
Student name:BBB
Student Address:Hyd
Branch Id  :B-111
Branch Name:CS
Student Details
----------------------
Student Id :S-333
Student name:CCC
Student Address:Hyd
Branch Id  :B-111
Branch Name:CS
```

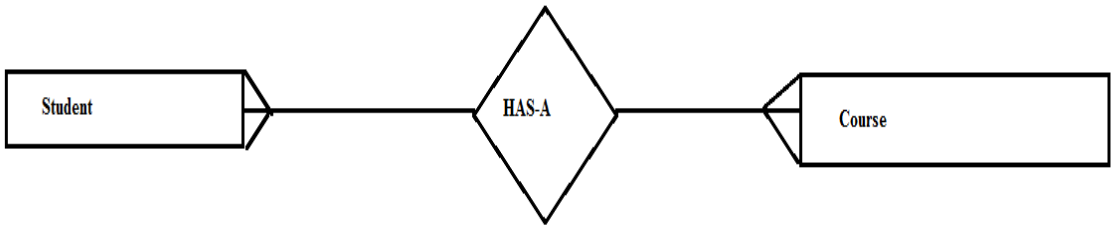
Data Flow Diagram on Many-To-One:



4.Many-To-Many Associations:

It is a relationship between entities,Where multiple instances of an entity should be mapped with multiple instances of another entity.

Ex:Multiple Students Have Joined with Multiple Courses.



Example programme on Many-To-Many:

```
class Course{
String cid;
String cname;
int ccost;
Course(String cid,String cname,int ccost){
this.cid=cid;
this.cname=cname;
this.ccost=ccost;
```

```
}}
class Student{
String sid;
String sname;
String saddr;
Course[] crs;
Student(String sid,String sname,String saddr,Course[] crs){
this.sid=sid;
this.sname=sname;
this.saddr=saddr;
this.crs=crs;
}
public void getStudentDetails(){
System.out.println("Student Details");
System.out.println("----------------");
System.out.println("Student Id   :"+sid);
System.out.println("Student name :"+sname);
System.out.println("Student Address:"+saddr);
System.out.println("CID  CNAME CCOST");
System.out.println("----------------------");
for(int i=0;i<crs.length;i++){
Course c=crs[i];
System.out.println(c.cid+" "+c.cname+" "+c.ccost);
}
System.out.println();
}}
```

```
class ManyToManyEx{
public static void main(String[] args){
Course c1=new Course("C-111","C",500);
Course c2=new Course("C-222","C++",1000);
Course c3=new Course("C-333","JAVA",5000);
Course[] crs=new Course[3];
crs[0]=c1;
crs[1]=c2;
crs[2]=c3;
Student std1=new Student("S-111","AAA","Hyd",crs);
Student std2=new Student("S-222","BBB","Hyd",crs);
Student std3=new Student("S-333","CCC","Hyd",crs);
std1.getStudentDetails();
std2.getStudentDetails();
std3.getStudentDetails();
}}
```


Output:

```
Student Details
----------------------
Student Id      :S-111
Student name :AAA
Student Address:Hyd
CID    CNAME  CCOST
-----------------------------------
C-111      C          500
C-222      C++      1000
C-333      JAVA     5000


Student  Details
-----------------------
Student Id   :S-222
Student name:BBB
Student Address:Hyd
CID    CNAME  CCOST
-----------------------------------
C-111      C          500
C-222     C++      1000
C-333     JAVA     5000


Student  Details
-----------------------
Student Id  :S-333
Student name:CCC
```

Student Address:Hyd

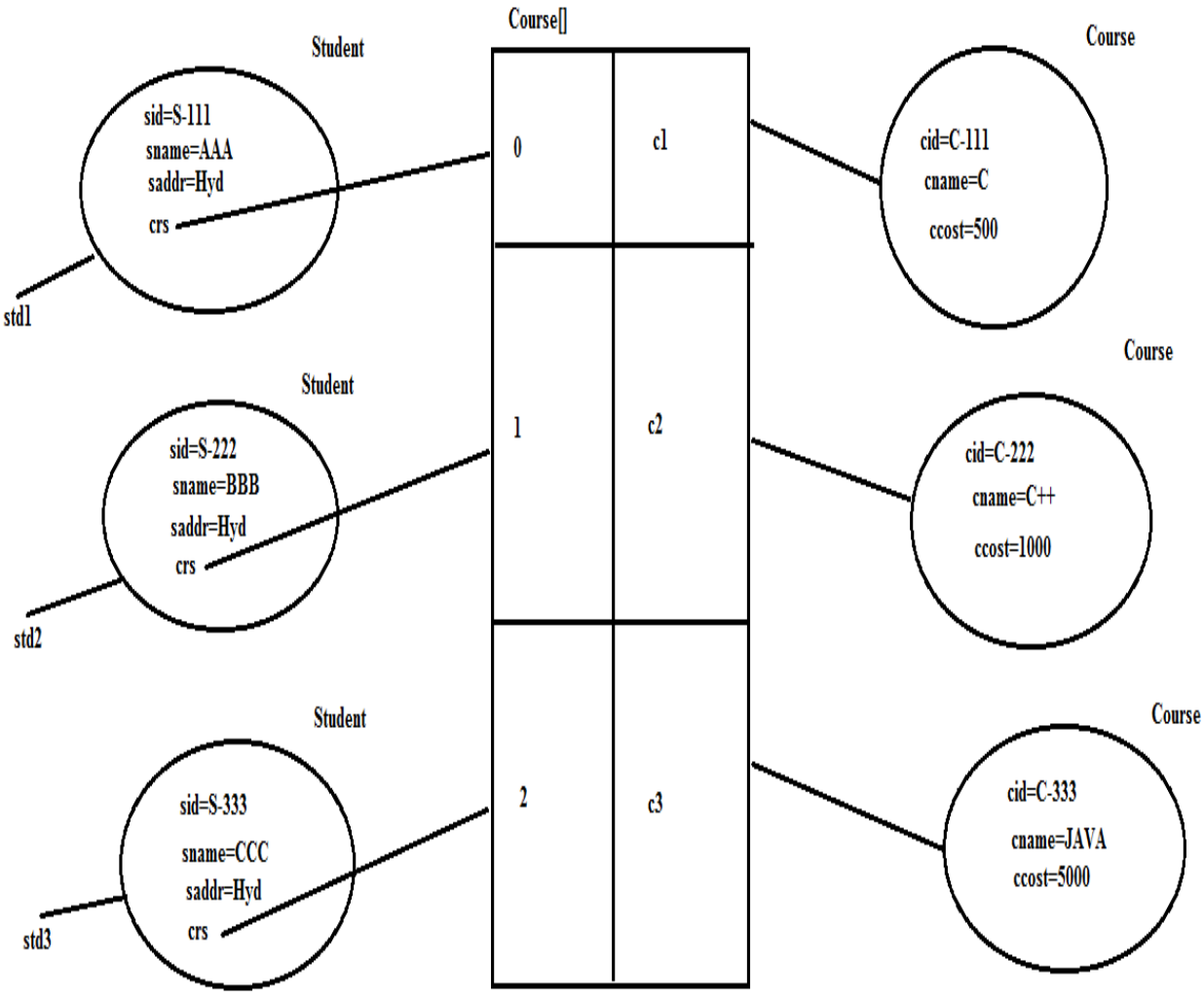
CID CNAME CCOST

C-111 C 500

C-222 C++ 1000

C-333 JAVA 5000

Data Flow Diagram on Many-To-Many:



Uses a RelationShip:

This is a relationship between entities,where one entity will use another entity upto a particular action or behaviour or method.

To provide USES-A Relationship in java applications,we have to declare contained Entity class reference variable as parameter to a method in Container entity class,not as class level variable.

If we declared contained entity reference variable as class level reference variable in container entity class then that relationship is "HAS-A" relationship.

Example programme on Uses a RelationShip:

```
class Account{

String accno;

String accName;

String accType;

int bal=10000;

Account(String accNo,String accName,String accType){

this.accNo=accNo;

this.accName=accName;

this.accType=accType;

}}

class Transaction{

String tx_tid;

String tx_Type;

Transaction(String tx_id,String tx_Type){

this.tx_Id=tx_Id;

this.tx_Type=tx_Type;

}

public void deposit(Account acc,int dep_Amt){
```

```
int initial_Amt=acc.bal;
int total_Avl_Amt=initial_Amt+dep_Amt;
acc.bal=total_Avl_Amt;
System.out.println("Transaction Details");
System.out.println("--------------------");
System.out.println("Transaction Id  :"+tx_Id);
System.out.println("Account Number :"+acc.accNo);
System.out.println("Account Type  :"+acc.accType);
System.out.println("Initial Amount :"+initial_Amt);
System.out.println("Deposit Amount :"+dep_Amt);
System.out.println("Total Avl Amount :"+total_Avl_Amt);
System.out.println("Transaction Status:SUCCESS");
System.out.println("********THANKQ,VISIT AGAIN**********");
}}
```



```
class UsesAEx{
public static void main(String[] args){
Account acc=new Account("abc123","Durga","Savings");
Transaction tx=new Transaction("T-111","Deposit");
tx.deposit(acc,5000);
}}
```

Output:

```
Transaction Details
---------------------------
Transaction  Id :T-111
Account Number :abc123
Account Type :Savings
Initial Amount :10000
Deposit Amount :5000
Total Avl Amount :15000
Transaction Status:SUCCESS
********THANKQ,VISIT AGAIN**********
```

Q)What is the difference between Aggregation and Composition?

Ans:In Object Orientation,both Aggregation and Composition are two level of associations or representations of associations.

Where Aggregation is representing weak association that is less dependency but composition is representing strong association that is more dependency.

In any Association,if contained object is existed even without container object,then this association is called as Weak Association and it is representing Aggregation.

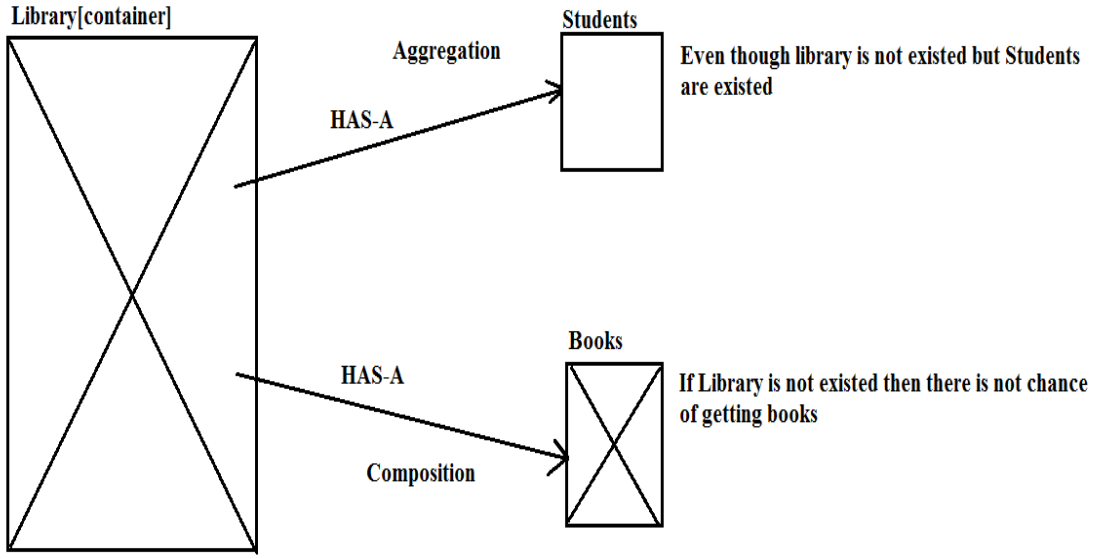
In any Association,if contained object is not existed without container object,then this association is called as Strong Association and it is representing Composition.

Ex:

If we take an association between "Library" and "Students","Library" and "Books".

"Students" can exist without "Library",so that,the association between "Library" and "Student" is Aggregation."Books" can not exist without "Library",so that,the association between "Library" and "Books" is composition

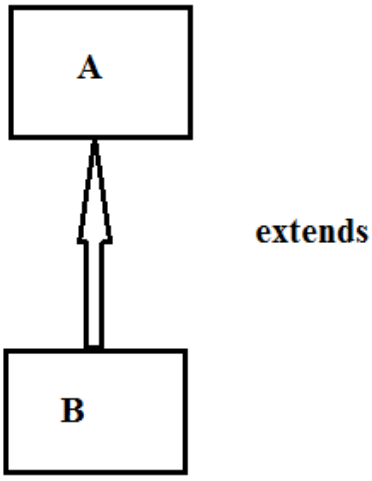
In the case of Aggregation,the life of the contained Object is independent of the container Object life.In the case of composition,the life of contained objects is depending on the container object life that is same as container object life.



IS-A Relationship:

It is a relationship between entity classes,it will provide inheritance between entity classes,where inheritance relationship will improve "code reusability" in Java application.

The process of getting variables and methods from one class to another class is called as Inheritance.



At the basic level of Object Orientation,There are two types of inheritances at basic level of Object orientation.

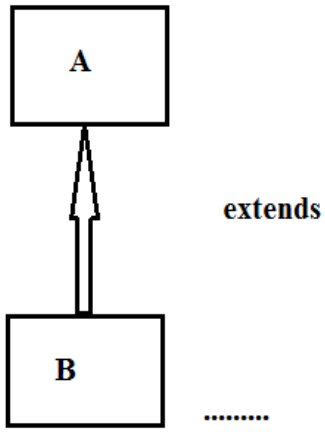
1.Single Inheritance

2.Multiple Inheritance

1.Single Inheritance:

The process of getting variables and methods from only one super class to one or more no.of subclasses is called as Single Inheritance.

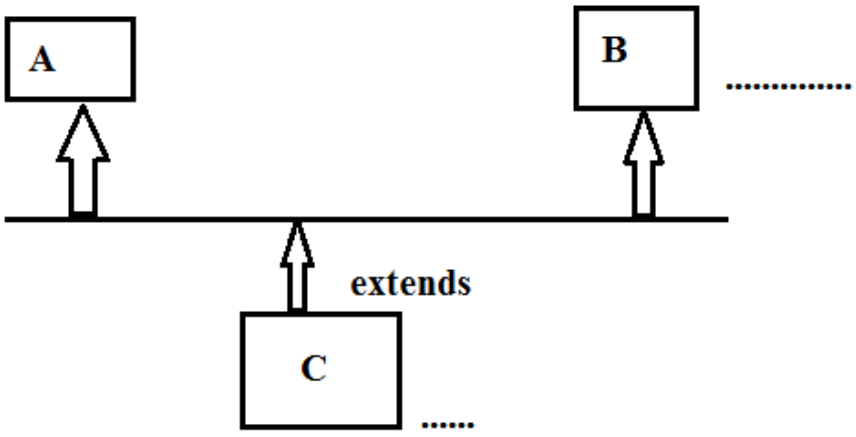
Java is able to allow Single Inheritance.



2.Multiple Inheritance:

The process of getting variables and methods from more than one super class to one or more no.of sub classes is called as Multiple Inheritance.

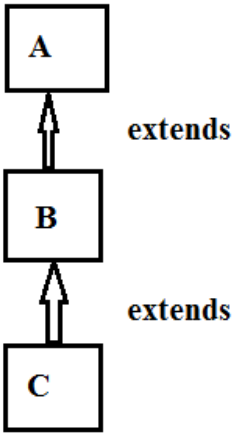
Java is not allowing Multiple Inheritance.



On the basis of Single & Multiple,there are 3 more Inheritances.
1. Multi-Level Inheritance
2. Hierarchial Inheritance
3. Hybrid Inheritance

1. Multi-Level Inheritance :

It is a Combination of single inheritance in more than one level.
Java is able to allow multi-level inheritance.



2. Hierarchial Inheritance :

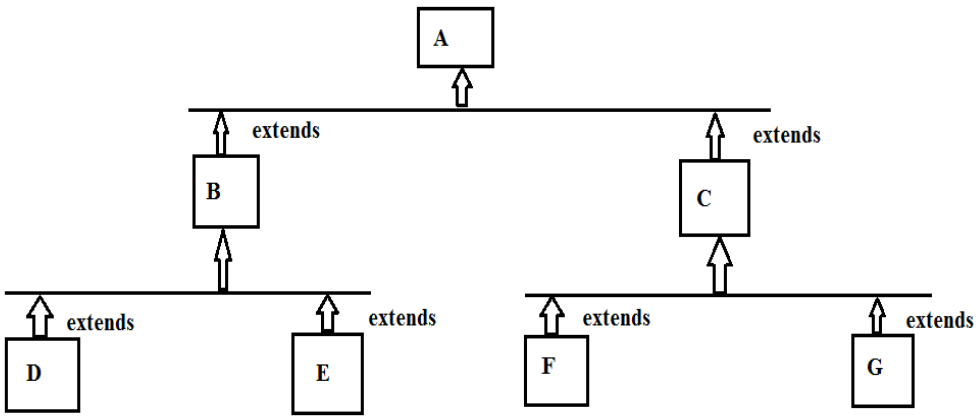
It is the combination of single inheritance in a particular structure.Java is able to allow

Hierarchical inheritance.

Ex: class A class B extends A class C extends A
{ { {
---------- ---------- ----------
---------- ---------- ----------
} } }

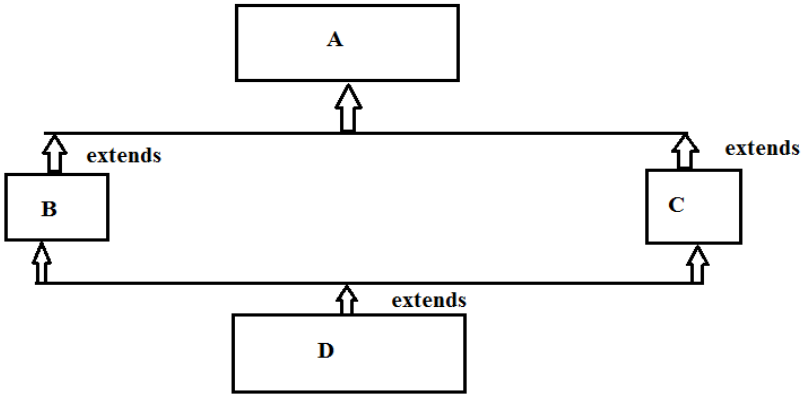
class D extends B class E extends B
{ {
---------- ----------
---------- ----------
} }

class F extends C class G extends C
{ {
---------- ----------
---------- ----------
} }



3. Hybrid Inheritance :

It is the combination of single inheritance and multiple inheritance.Java is not allowing Hybrid Inheritance.



Example programme for Single Inheritance:

```
class Employee
{
String eid;
String ename;
String eaddr;
```

```
public void getEmpDetails()
{
System.out.println("Employee Id :"+eid);
System.out.println("Employee name :"+ename);
System.out.println("Employee Address :"+eaddr);
}
}
class Manager extends Employee
{
Manager(String eid1,String ename1,String eaddr1)
{
eid=eid1;
ename=ename1;
ename=eaddr1;
}
public void getManagerDetails()
{
System.out.println("manager Details");
System.out.println("-----------------");
getEmpDetails();
}
}
class Accountant extends Employee
{
Accountant (String eid1,String ename1,String eaddr1)
{
eid=eid1;
ename=ename1;
eaddr=eaddr1;
}
public void getAccountantDetails()
{
System.out.println("Accountant Details");
System.out.println("--------------------");
getEmpDetails():
}
}
class InheritanceEx
{
public static void main(String[] args)
{
Manager m=new Manager("E-111","AAA","Hyd");
m.get ManagerDetails();
System.out.println();
```

```
Accountant acc=new Accountant("E-222","BBB","Hyd");
acc.getAccountantDetails();
}
}
```

On Command Prompt:

```
==================
D:\java7>javac Inheritance Ex.java
D:\java7>java Inheritance Ex

Manager Details:
===============
Employee Id :E-111
Employee Name :AAA
Employee Address :Hyd

Accountant Details:
================
Employee Id :E-222
Employee Name :BBB
Employee Address :Hyd
```

Static Context in Inheritance:

- In the case of inheritance,if we create an object for sub class then JVM has to execute sub class constructor,before executing sub class constructor,JVM has to check whether sub class bytecode is loaded already in the memory or not,if not,JVM has to load subclass bytecode to the memory.
- In this context,before loading sub class bytecode to the memory,first,JVM has to load the respective super class bytecode to the memory.

- Therefore,in the case of inheritance,JVM will load all the classes bytecode right from super class to sub class.
- If we provide static context in both super class and subclass then JVM will recognize and execute static context of the respective classes at the time of loading the respective classes.

Example programme on static Context Inheritance:

```
class A{

static{

System.out.println("SB-A");

}}

class B extends A{

static{

System.out.println("B-con");

}}

class C extends B{

static{

System.out.println("SB-C");

}}

class Test{

public static void main(String[] args){

C c=new C();

}}
```

Ouput:

```
SB-A

B-con

SB-C

==========================
```

```
class A{
static{
System.out.println("SB-A");
}
static int m1(){
System.out.println("m1-A");
return 10;
}
static int i=m1();
}
class B extends A{
static int j=m2();
static{
System.out.println("SB-B");
}
static int m2(){
System.out.println("m2-B");
return 20;
}}
class C extends B{
static int m3(){
System.out.println("m3-C");
return 30;
}
static int k=m3();
static{
```

```
System.out.println("SB-C");
}}
class Test{
public static void main(String[] args){
C c1=new C();
C c2=new C();
}}
```

O/P:SB-A

m1-A

m2-B

SB-B

m3-C

SB-C

Instance Context in Inheritance:

- In the case of Inheritance,if we create object for sub class then JVM has to execute sub class constructor,but before executing sub class constructor JVM has to execute 0-argument constructor in the respective super class.

Example programmes on Instance Context in Inheritance:

```
class A{
A(){
System.out.println("A-con");
}}
class B extends A{
B(){
System.out.println("B-con");
}}
class C extends B{
C(){
System.out.println("C-con");
}}
class Test{
public static void main(String[] args){
C c=new C();
}}
O/P:A-Con
      B-Con
      C-Con
================================
class A{
A(){
System.out.println("A-con");
}}
class B extends A{
}
```

```
class C extends B{
C(){
System.out.println("C-con");
}}

class Test{
public static void main(String[] args){
C c=new C();
}}
```

O/P:A-Con

C-Con

==========

```
class A{
A(int i){
System.out.println("A-int-param-con");
}}
class B extends A{
 B(int i){
 System.out.println("B-int-param-con");
 }}
 class C extends B{
 C(int i){
 System.out.println("C-int-param-con");
 }}
 class Test{
 public static void main(String[] args){
```

```
C c=new C(10);
}}
```

Status:Compilation Error

Reason:In case of Inheritance,super classes must have 0-argument constructors irrespective of the sub class constructors.

- In the case of Inheritance,if we provide instance context at all the classes then JVM will recognize and execute them just before executing the respective class constructors.

```
class A{
A(){
System.out.println("A-con");
}
int i=m1();
int m1(){
System.out.println("m1-A");
return 10;
}
{
System.out.println("IB-A");
}}
class B extends A{
int j=m2();
int m2(){
System.out.println("m2-B");
return 20;
}
```

```
{
System.out.println("IB-B");
}
```


```
B(){
System.out.println("B-Con");
}}
class C extends B{
C(){
System.out.println("C-con");
}
{
System.out.println("IB-C");
}
int k=m3();
int m3(){
System.out.println("m3-C");
return 30;
}}
```

```
class Test{
public static void main(String args[]){
C c=new C();
}}
```

Ouput:

```
m1-A
IB-A
A-con
m2-B
IB-B
B-Con
IB-C
m3-C
c-con
```

```
class A{
A(){
System.out.println("A-con");
}
static{
System.out.println("SB-A");
}
int m1(){
System.out.println("m1-A");
return 10;
}
```

```
static int m2(){
System.out.println("m2-A");
return 20;
}
{
System.out.println("IB-A");
}
static int i=m2();
int j=m1();
}
class B extends A{
{
System.out.println("IB-B");
}
int m3(){
System.out.println("m3-B");
return 30;
}
static{
System.out.println("SB-B");
}
int k=m3();
B(){
System.out.println("B-Con");
}
static int l=m4();
```

```
static int m4(){
System.out.println("m4-B");
return 40;
}}
class C extends B{
static int m5(){
System.out.println("m5-C");
return 50;
}
int m6(){
System.out.println("m6-C");
return 60;
}

C(){
System.out.println("C-con");
}
int m6();
static int n=m5();{
System.out.println("IB-C");
}
static{
System.out.println("SB-C");
}
}
class Test{
```

```
public static void main(String args[]){
C c1=new C();
C c2=new C();
}}
```

```
O/P:  SB-A          A-con          m1-A
      m2-A          IB-B           A-con
      SB-B          m3-B           IB-B
      m4-B          B-con          m3-B
      m5-C          m6-C           B-con
      SB-C          IB-C           m6-C
      IB-A          C-con          IB-C
      m1-A          IB-A           C-con
```


Super Keyword:

- Super is a Java keyword,it can be used to represent super class object from sub classes.

There are three ways to utilize "super" keyword.

1.To refer super class variables

2.To refer super class constructors

3.To refer super class methods.

1.To refer super class variables:

- If we want to refer super class variables,by using 'super' keyword then we have to use the following syntax.

  super.var_Name;

  NOTE:We will utilize super keyword,to access super class variables when we have same set of variables at local,at current class and at the super class.

  Example programme,To refer super class variables:

```
class A{
int i=100;
int j=200;
}
class B extends A{
int i=10;
int j=20;
B(int i,int j){
System.out.println(i+"    "+j);
System.out.println(this.i+"    "+this.j);
```

```
System.out.println(super.i+"    "+super.i);
}}

class Test{
public static void main(String args[]){
B b=new B(50,60);
}}
```

Output:

```
50      60
10      20
100     100
```


2.To refer super class constructors:

- If we want to refer super class constructor from sub class by using 'super' keyword then we have to use the following syntax.

```
super([Param_List]);
```

NOTE: In general,in inheritance,JVM will execute 0-argument constructor before executing sub class constructor.In this context,instead of executing 0-argument constructor we want to execute a parametrized constructor at super class,for this,we have to use 'super' keyword.

Example programme,To refer super class constructors:

```
class A{
```

```
A(){
System.out.println("A-Con");
}
A(int i){
System.out.println("A-int-param-Con");
}}
class B extends A{


B(){
super(10);
System.out.println("B-Con");
}}
class Test{
public static void main(String args[]){
B b=new B();
}}
```

Output:

```
A-int-param-Con
B-Con
```

- If we want to access super class constructor from subclass by using "super" keyword then the respective "super" statement must be provided as first statement.
- If we want to access super class constructor from sub class by using "super" keyword then the respective "super" statement must be provided in the subclass constructors only,not in subclass normal Java methods.
- If we violate any of the above conditions then compiler will rise an error like "call to super must be first statement in constructor".

NOTE:Due to the above rules and regulations,it is not possible to access more than one super class constructor from a single sub class constructor by using "super" keyword.

- In the case of inheritance, when we access sub class constructor then JVM will execute super class 0-arg constructor then sub class constructor, this flow of execution is possible in Java because of the following compiler actions over source code at the time of compilation.

  a)Compiler will goto each and every class available in the source file and checks whether any requirement to provide "default constructors".

  b)If any class is identified with out user defined constructor explicitly then compiler will add a 0-argument constructor as default constructor.

  c)After providing default constructors,compiler will goto all the constructors at each and every class and compiler will check "super" statement is provided or not by the developer explicitly to access super class constructor.

  d)If any class constructor is identified with out "super" statement explicitly then compiler will append "super()" statement in the respective constructor to access a 0-argument constructor in the respective super class.

  e)With the above compiler actions,JVM will execute super class 0-arg constructor as part of executing sub class constructor explicitly.

```
class A{
A(){
System.out.println("A-con");
}
}
class B extends A
{
B(){
System.out.println("B-con");
}}
class C extends B{
C(){
System.out.println("C-con");
}}
class Test{
public static void main(String[] args){
C c=new C();
}}
```

compile

```
class Test{
Test(){ super(); }
public static void main(String[] args)
{
C c=new C();
}}
```

JVM will execute the object class 0-arg constructor

```
class A{
A(){
super();
System.out.println("A-con");
}}
class B extends A
{
B(){
super();
System.out.println("B-con");
}}
class C extends B{
C(){
super();
System.out.println("C-con");
}}
```

Write the translated code:

```
class A{

A(){

System.out.println("A-con");
```

```
}}
class B extends A{
B(int i){
System.out.println("B-int-param-con");
}}
class C extends B{
C(int i){
System.out.println("C-int-param-con");
}}
class Test{
public static void main(String args[]){
C c=new C();
}}
```

Status:Compilation Error

==========

```
class A{
A(){
System.out.println("A-con");
}}
class B extends A{
B(int i){
System.out.println("B-int-param-con");
}}
class C extends B{
C(int i){
super(10);
```

```
System.out.println("C-int-param-con");
}}
class Test{
public static void main(String args[]){
C c=new C(10);
}}
```

Output:

```
A-con
B-int-param-con
C-int-param-con
```


3.To refer super class method:

- If we want to refer super class methods from sub class by using "super" keyword then we have to use the following syntax.

  super.method_Name([Param_List]);

  NOTE:In Java applications,when we have same method at both subclass and at super class,if we access that method at sub class then JVM will execute sub class method only,not super class method because JVM will give more priority for the local class methods.In this context,if we want to refer super class method over sub class method then we have to "super" keyword.

  Example Programme to refer super class method:

```
class A{
void m1(){
```

```
System.out.println("m1-A");
}}
class B extends A{
void m2(){
System.out.println("m2-B");
m1();
this.m1();
super.m1();
}}
void m1(){
System.out.println("m1-B");
}}
class Test{
public static void main(String args[]){
B b=new B();
b.m2();
}}
```

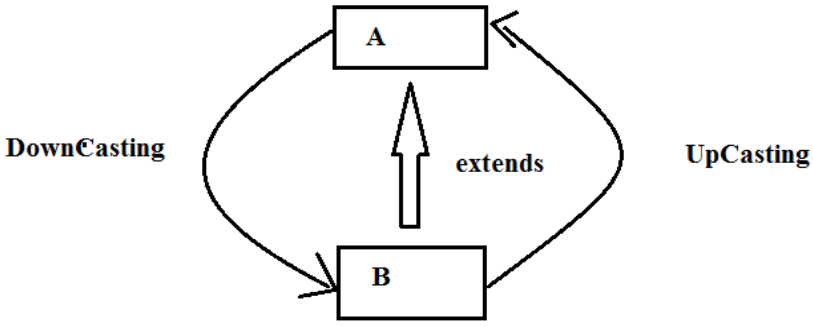

Class Level Type Casting:

- The process of converting the data from one user defined data type to another user defined data type is called as User Defined Data type casting or Class level type Casting.
- If we want to perform Class Level Type Casting or User Defined data types casting then we must require either "extends" or "implements" relationship between two user defined data types.

There are two types of User defined data type casting:

1.UpCasting.

2.DownCasting.



1.UpCasting:

- The process of converting the data from sub type to super type is called as UpCasting.
- To perform Upcasting,we have to assign sub class reference variable to super class reference variable.

Ex:

```
class A{

}

class B extends A{

}

B b=new B();

A a=b;
```

- If we compile the above code then compiler will check whether 'b' variable data type is compatible with 'a' variable data type or not,if not,compiler will rise an error like "InCompatible Types".If "b" variable data type id compatible to "a" variable data type then compiler will not rise any error.

NOTE:In Java applications,always sub class types are compatible with super class types,so that we can assign sub class reference variable to super class reference variables directly.

NOTE:In Java,super class types are not compatible with sub class types,so that,we cannot assign super class reference variables to sub class reference variables directly,where if we want to assign super class reference variables to sub class reference variables then we must require "cast operator" explicitly,that is called as "Explicit Type Casting".

If we execute the above code then JVM will perform the following two actions.

1.JVM will convert "b" variable data type[sub class type] to "a" variable data type[Super class type] implicitly.

2.JVM will copy the reference value of "b" variable to "a" variable.

With the upcasting,we are able to access only super class members among the availability of both super class and sub class members in sub class object.

```
class A{
void m1(){
System.out.println("m1-A");
}}
class B extends A{
void m2(){
System.out.println("m2-B");
}}
class Test{
public static void main(String[] args){
B b=new B();
b.m1();
b.m2();
A a=b;
a.m1();
//a.m2();---->error
}}
```

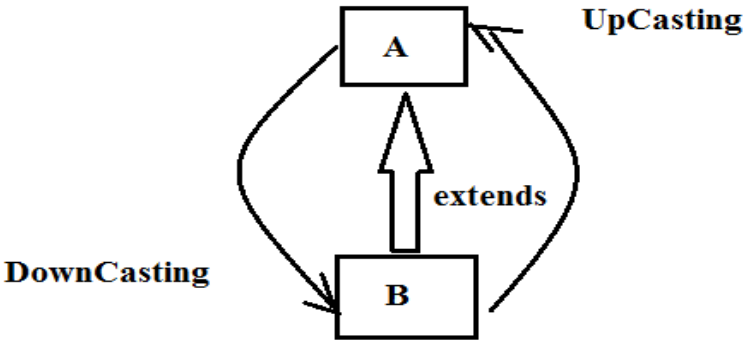
Output:

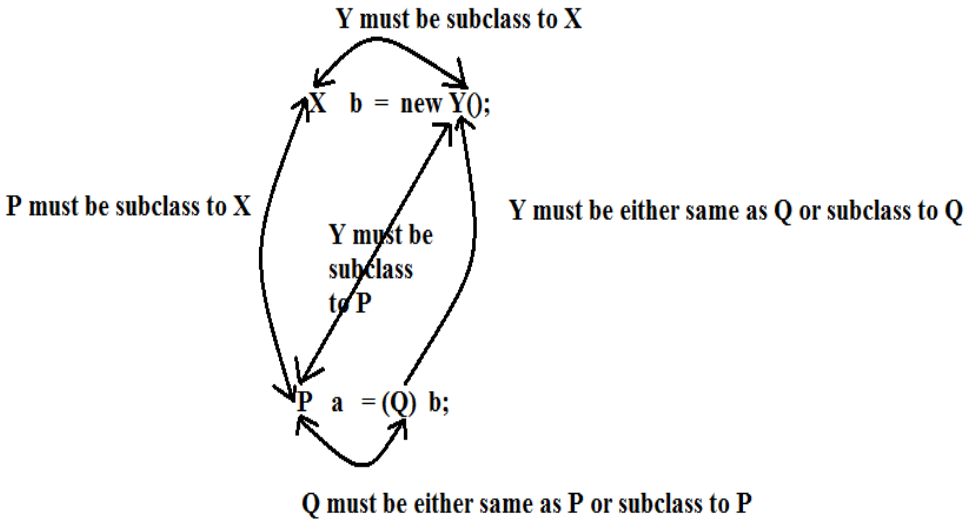
```
m1-A
m2-B
m1-A
```

2.DownCasting:

- The process of converting the data from super type to sub type is called as "DownCasting".

  If we want to perform DownCasting then we have to use the following format.






```
class A{
void m1(){
System.out.println("m1-A");
}}
class B extends A{
void m2(){
System.out.println("m2-B");
}}
class Test{
public static void main(String args[]){
case 1:
  A a=new A();
  B b=a;
```

Status:Compilation error,incompatible types

Reason:In Java,always,subclass types are compatible with super class types but super class types are not compatible with sub class types.It is possible to assign sub class reference variables directly but it is not possible to assign super class reference variable to sub class reference variables directly,if we assign then compiler will rise an error like "InCompatible Types error".

Case 2:

```
A a=new A();

B b=(B)a;
```

Status:No Compilation error,but classCastException

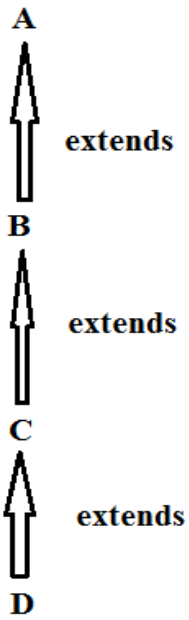
Reason:In Java,always,it is possible to keep subclass object reference value in super class reference variable but it is not possible to keep superclass object reference value in subclass reference variable.If we are trying to keep super class object reference value in sub class reference variable then JVM will rise an exception like "java.lang.classCastException".

Case 3:

```
A a=new B();

B b=(B)a;
```

Status:No compilation error,no exception

```
b.m1();

b.m2();
```



Ex1:

A a=new A();

B b=a;

Status:Compilation error,incompatible types

Ex2:

A a=new A();

B b=(B)a;

Status:No Compilation error,ClassCastException

Ex3:

A a=new A();

B b=(B)a;

Status:No Compilation error,No Exception

Ex4:

A a=new C();

B b=(C)a;

Status:No Compilation error,No Exception

Ex5:

A a=new B();

B b=(C)a;

Status:No Compilation Error;ClassCastException

Ex6:

A a=new C();

C c=(D)a;

Status:No Compilation Error,ClassCastException

Ex7:

B b=new D();

C c=(D)b;

Status:NO Compilation Error,No Exception

Ex8:

A a=new D();

D d=(D)(C)(B)a;

Status:No Compilation error,No Exception

Ex9:

A a=new C();

D d=(D)(C)(B)a;

Status:No Compilation Error,ClassCastException

Ex10:

A a=new C();

C c=(D)(C)(B)a;

Status:No Compilation Error,ClassCastException

Polymorphism:

Polymorphism is a Greek word,where poly means many and morphism means Structures.

If one thing is existed in more than one form then it is called as Polymorphism.

There are two types of Polymorphisms

1.Static Polymorphism or Early binding

2.Dynamic Polymorphism or Late Binding

1.Static Polymorphism:

If the Polymorphism is existed at compilation time then that Polymorphism is called as Static Polymorphism.

Ex:Method Overloading

2. Dynamic Polymorphism:

If the Polymorphism is existed at runtime then that Polymorphism is called as Dynamic Polymorphism

Ex:Method Overriding.

Method Overloading:

The process of extending the existed method functionality upto some new Functionality is called as Method Overloading.

If we declare more than one method with the same name and with the different parameter list is called as Method Overloading.

To perform method overloading, we have to declare more than one method with different method signatures that is same method name and different parameter list.

Example Program me on Method Overloading:

```
class A{

void add(int i,int j){

System.out.println(i+j);

}

void add(float f1,float f2){

System.out.println(f1+f2);

}

void add(String str1,String str2){

System.out.println(str1+str2);
```

```
}}
class Test{
public static void main(String[] args){
A a=new A();
a.add(10,20);
a.add(22.22f,33.33f);
a.add("abc","def");
}}
```

Output:

```
30
55.550003
abcdef
```

```
class Employee{
void gen_Salary(int basic,float hk,float pf,int ta){
float salary=basic+((basic*hk)/100)-((basic*pf)/100)+ta);
System.out.println("Salary :"+salary);
}
void gen_Salary(int basic,float hk,float pf,int ta,int bonus){
float salary=basic+((basic*hk)/100)-((basic*pf)/100)+ta)+bonus;
System.out.println("Salary :"+salary);
}}
class Test{
public static void main(String[] args){
Employee e=new Employee();
e.gen_Salary(20000,25.of,12.0f,2000);
```

```
e.gen_Salary(20000,25.of,12.0f,2000,5000);

}}
```

Method Overriding:

The process of replacing existed method functionality with some new functionality is called as Method Overriding.

To perform Method Overriding, we must have inheritance relationship classes.

In Java applications, we will override super class method with sub class method.

In Java applications,we will override super class method with subclass method.

If we want to override super class method with sub class method then both super class method and sub class method must have same method prototype.

Steps to perform Method Overriding:

1.Declare a super class with a method.

2.Declare a sub class and provide the same super class method with different implementation.

3.In main class,in main() method,prepare object for sub class and prepare reference variable for super class[UpCasting].

4.Access super class method then we will get output from sub class method.

Example Programme on Mehhod Overriding:

```
class Loan{
```

```
public float getIR(){
return 7.0f;
}
}
class GoldLoan extends Loan{
public float getIR(){
return 10.5f;
}
}
class StudyLoan extends Loan{
public float getIR(){
return 12.0f;
}}

class CraftLoan extends Loan{
}
class Test{
public static void main(String[] args){
Loan gold_Loan=new GoldLoan();
System.out.println("Gold Loan IR  :"+gold_Loan.getIR()+"%");
Loan study_Loan=new StudyLoan();
System.out.println("Study Loan IR :"+study_Loan.getIR()+"%");
Loan craft_Loan=new CraftLoan();
System.out.println("Craft Loan IR  :"+craft_Loan.getIR()+"%");
}}
```

NOTE: To prove method overriding in Java, we have to access super class method but JVM will execute the respective sub class method and JVM has to provide

output from the respective sub class method, not form super class method.To achieve the above requirement we must create reference variable for only super class and we must create object for sub class.

```
class A{

void m1(){

System.out.println("m1-A");

}}

class B extends A{

void m1(){

System.out.println("m1-B");

}}

class Test{

public static void main(String args[]){

/* A a=new A();

  a.m1();

  Status:Here Method Overriding is not happend,because Mehtod overriding must require sub class

object,not super class object.

*/

/*

B b=new B();

b.m1();

Status:Here method Overriding is happened,but,to prove method overriding we must require superclass reference variable,not sub class reference variable,because,subclass reference variableis able to access only sub class method when we have the same method in both sub class andsuper class.

*/


A a=new B();
```

```
a.m1();

}}
```

Rules to perform Method Overriding:

1.To override super class method with sub class then super class method must not be declared asprivate.

Ex:

```
class A{

private void m1(){

System.out.println("m1-A");

}}

class B extends A{

void m1(){

System.ou.println("m1-B");

}}

class Test{

public static void main(String args[]){

A a=new A();

a.m1();

}}
```

2.To override super class method with sub class method then sub class method should have the samereturn type of the super class method.

Ex:

```
class A{

int m1(){

System.ou.println("m1-A");

return 10;
```

```
}}

class B extends A{

void m1(){

System.out.println("m1-B");

}}

class Test{

public static void main(String args[]){

A a=new B();

a.m1();

}}
```

3.To override super class method with sub class method then super class method must not be declared as final sub class method may or may not be final.

Ex:

```
class A{

void m1(){

System.out.println("m1-A");

}}

class B extends A{

final void m1(){

System.out.println("m1-B");

}}

class Test{

public static void main(String[] args){

A a=new B();

a.m1();

}}
```

4.To override superclass method with sub class method either super class method or subclass method as static then compiler will rise an error.If we declare both super and sub class method as static in method overriding compiler will not rise any error,JVM will provide output from the super class method.

NOTE:If we are trying to override superclass static method with sub class static method then super class static method will override subclass static method,where JVM will generate output from super class static method.

Example:

```
class A{

static void m1(){

System.out.println("m1-A");

}}

class B extends A{

static void m1(){

System.out.println("m1-B");

}}

class Test{

public static void main(String args[]){

A a=new B();

a.m1();

}}
```

5.To override super class method with subclass method,sub class method must have either same scope of the super class method or more scope when compared with super class method scope otherwise compiler will rise an error.

Ex:

```
class A{

protected void m1(){

System.out.println("m1-A");

}}
```

```
class B extends A{

public void m1(){

System.out.println("m1-B");

}}

class Test{

public static void main(String args[]){

A a=new A();

a.m1();

}}
```

6.To override super class method with subclass method subclass method should have either same access privileges or weaker access privileges when compared with super class method access privileges.

Q)What are the differences between method overloading and method overriding?

Ans:

Method Overloading	Method Overriding
1. The process of extending the existed method functionality with new functionality is called as Method Overloading.	The process of replacing existed method functionality with new functionality is called as Method Overriding
2.In the case of method overloading, different method signatures must be provided to the methods	In the case of method overriding same method prototypes must be provided to the methods.

3.With or without inheritance we can perform method overloading	With inheritance only we can perform Method overriding

Consider the following programme:

```
class DB_Driver{

public void getDriver(){

System.out.println("Type-1 Driver");

}}

class New_DB_Driver extends DB_Driver{

public void getDriver(){

System.out.println("Type-4-Driver");

}}


class Test{

public static void main(String args[]){

DB_Driver driver=new New_DB_Driver();

driver.getDriver();

}}
```

In the above example, method overriding is implemented, in method overriding, for super class method call JVM has to execute subclass method, not super class method. In method overriding, always JVM is executing only subclass method, not super class method. In method overriding, it is not suggestible to manage super class method body without execution, so that, we have to remove super class method body as part of code optimization. In Java applications, if we want to declare a method without body then we must declare that method as "Abstract Method".

If we want to declare abstract methods then the respective class must be abstract class.

```
abstract class DB_Driver{

public abstract void getDriver();

}
```

```
class New_DB_Driver extends DB_Driver{

public void getDriver(){

System.out.println("Type-4 Driver");

}}

class Test{

public static void main(String args[]){

DB_Driver driver=new New_DB_Driver();

driver.getDriver();

}}
```

In Java applications, if we declare any abstract class with abstract methods, then it is convention to implement all the abstract methods by taking sub class.

To access the abstract class members, we have to create object for subclass and we have to create reference variable either for abstract class or for subclass.

If we create reference variable for abstract class then we are able to access only abstract class members, we are unable to access subclass own members

If we declare reference variable for subclass then we are able to access both abstract class members and subclass members.

```
abstract class A{

void m1(){

System.out.println("m1-A");

}

abstract void m2();

abstract void m3();

}

class B extends A{

void m2(){

System.out.println("m2-B");

}
```

```
void m3(){
System.out.println("m3-B");
}
void m4(){
System.out.println("m4-B");
}}
class Test{
public static void main(String args[]){
A a=new B();
a.m1();
a.m2();
a.m3();
//a.m4();---error
B b=new B();
b.m1();
b.m2();
b.m3();
b.m4();
}}
```

In Java applications,it is not possible to create Object fro abstract classes but it possible to provide constructors in abstract classes,because,to recognize abstract class instance variables in order to store in the sub class objects.

```
abstract class A{
A(){
System.out.println("A-Con");
}}
class B extends A{
```

```
B(){
System.out.println("B-Con");
}}
class Test{
public static void main(String[] args){
B b=new B();
}}
```

In Java applications, if we declare any abstract class with abstract methods then it is mandatory to implement all the abstract methods in the respective subclass. If we implement only some of the abstract methods in the respective subclass then compiler will rise an error, where to come out from compilation error we have to declare the respective subclass as an abstract class and we have to provide implementation for the remaining abstract methods by taking another subclass in multilevel inheritance.

```
abstract class A{
abstract void m1();
abstract void m2();
abstract void m3();
}
abstract class B extends A{
void m1(){
System.out.println("m1-A");
}}
class C extends B{
void m2(){
System.out.println("m2-C");
}
void m3(){
```

```
System.out.println("m3-C");
}}
class Test{
public static void main(String[] args){
A a=new C();
a.m1();
a.m2();
a.m3();
}}
```


In Java applications, if we want to declare an abstract class then it is not at all mandatory condition to have atleast one abstract method, it is possible to declare abstract class without having abstract methods but if we want to declare a method as an abstract method then the respective class must be abstract class.

```
abstract class A{
void m1(){
System.out.println("m1-A");
}}
class B extends A{
void m2(){
System.out.println("m2-B");
```

```
}}

class Test{

public static void main(String args[]){

A a=new B();

a.m1();

//a.m2();---------->Error

B b=new B();

b.m1();

b.m2();

}}
```

In Java applications, it is possible to extends an abstract class to concrete class and from concrete class to abstract class.

```
class A{

void m1(){

System.out.println("m1-A");

}}
```


```
abstract class B extends A{

abstract void m2();
```

```
}
class C extends B{
void m2(){
System.out.println("m2-C");
}
void m3(){
System.out.println("m3-C");
}}
class Test{
public static void main(String args[]){
A a=new C();
a.m1();
//a.m2();--->error
//a.m3();--->error
B b=new C();
b.m1();
b.m2();
//b.m3();--->error
C c=new C();
c.m1();
c.m2();
c.m3();
}}
```

Note: In Java applications, it is not possible to extend a class to the same class, if we do the same then compiler will rise an error like "cyclic inheritance involving".

class A extends A{

}

Status: Compilation Error: “cyclic inheritance involving".

class A extends B{

}

class B extends A{

}

Status: Compilation Error: “cyclic inheritance involving".

Interfaces:

Interface is a Java Feature, it will allow only abstract methods.

In Java applications, for interfaces, we are able to create only reference variables, we are unable to create objects.

In the case of interfaces, by default, all the variables are "public static final".

In the case of interfaces, by default, all the methods are "public and abstract".

In Java applications, constructors are possible in classes and abstract classes but constructors are not possible in interfaces.

Interfaces will provide more sharability in Java applications when compared with classes and abstract classes.

In Java applications, if we declare any interface with abstract methods then it is convention to declare an implementation class for the interface and it is convention to provide implementation for all the abstract methods in implementation class.

Interface I{

```
void m1();
void m2();
void m3();
}
class A implements I{
public void m1(){
System.out.println("m1-A");
}
public void m2(){
System.out.println("m2-A");
}
public void m3(){
System.out.println("m3-A");
}}
public void m4(){
System.out.println("m4-A");
}}
class Test{
public static void main(String args[]){
I i=new A();
i.m1();
i.m2();
i.m3();
//i.m4();------>error
A a=new A();
a.m1();
```

```
a.m2();
a.m3();
a.m4();
}}
```

In Java applications, if we declare an interface with abstract methods then it is mandatory to provide implementation for all the abstract methods in the respective implementation class. In this context if we provide implementation for some of the abstract methods at the respective implementation class then compiler will rise an error, where to come out from the compilation error we have to declare the respective implementation class as an abstract class and we have to provide implementation for the remaining abstract methods by taking a sub class for the abstract class.

```
interface I{
void m1();
void m2();
void m3();
}
abstract class A implements I{
public void m1(){
System.out.println("m1-A");
}}
class B extends A{
public void m2(){
System.out.println("m2-B");
}
public void m3(){
System.out.println("m3-B");
}}
```

```
class Test{
public static void main(String args[]){
I i=new I();
i.m1();
i.m2();
i.m3();
A a=new B();
a.m1();
a.m2();
a.m3();
B b=new B();
b.m1();
b.m2();
b.m3();
}}
```


In Java applications, it is not possible to extend more than one class to a single class but it is possible to extend more than one interface to a single interface.

```
interface I1{
```

```
void m1();
}
interface I2{
void m2();
}
interface I3 extends I1,I2{
void m3();
}
class A implements I3{
public void m1(){
System.out.println("m1-A");
}


public void m2(){
System.out.println("m2-A");
}
public void m3(){
System.out.println("m3-A");
}}
class Test{
public static void main(String args[]){
I1 i1=new A();
i1.m1();
I2 i2=new A();
i2.m2();
```

```
I3 i3=new A();

i3.m1();

i3.m2();

i3.m3();

A a=new A();

a.m1();

a.m2();

a.m3();

}}
```

In Java applications, it is possible to implement more than one interface into a single implementation class.

```
interface I1{

void m1();

}


interface I2{

void m2();

}

interface I3{

void m3();

}

class A implements I1,I2,I3{

public void m1(){

System.out.println("m1-A");

}

public void m2(){

System.out.println("m2-A");
```

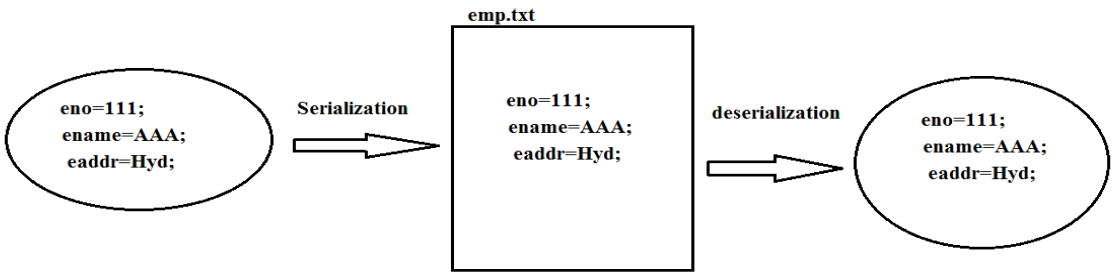
```
}
public void m3(){
System.out.println("m3-A");
}}
class Test{
public static void main(String args[]){
I1 i1=new A();
i1.m1();
I2 i2=new A();
i2.m2();
I3 i3=new A();
i3.m3();
A a=new A();
a.m1();
a.m2();
a.m3();
}}
```


Marker Interface is an Interface, it will not include any abstract method and it will provide some abilities to the objects at runtime of our Java application.

Ex: java.io.Serializable

The process of separating the data from an object is called as Serialization.

The process of reconstructing an Object on the basis of data is called as Deserialization.



Serialization & Deserialization

Where java.io.Serializable interface is a marker interface, it was not declared any method but it will make eligible any object for Serialization and Deserialization.

Valid Syntaxes between classes,abstract classes and Interfaces:

class extends class --->Valid

class extends class,class--->InValid

class extends abstract class--->Valid

class extends abstract class,class--->InValid

class extends abstract class,abstract class--->InValid

class extends interface--->InValid

class implements interface--->Valid

class implements interface,interface--->Valid

class implements interface extends class--->InValid

class implements interface extends abstract class--->InValid

class extends class implements interface--->Valid

class extends abstract class implements interface--->Valid

abstract class extends class--->Valid

abstract class extends abstract class--->Valid

abstract class extends class,class--->InValid

abstract class extends abstract class,abstract class--->InValid

abstract class extends class,abstract class--->InValid

abstract class extends interface--->InValid

abstract class implements interface--->Valid

abstract class implements interface,interface--->Valid

abstract class extends class implements interface--->Valid

abstract class extends abstract class implements interface--->Valid

abstract class implements interface extends class-->InValid

abstract class implements interface extends abstract class-->InValid

interface extends interface -->Valid

interface extends interface,interface -->Valid

interface extends class -->InValid

interface extends abstract class -->InValid

interface implements interface -->InValid

JAVA8 Features over Interfaces:

1.Default Methods in Interfaces

2.Static Methods in Interfaces

3.Functional Interfaces

1.Default Methods in Interfaces:

In general, if we declare abstract methods in an interface then we have to implement all that interface methods in more no.of classes with variable implementation part.

In the above context, if we require any method implementation common to every implementation class with fixed implementation then we have to implement that method in the interface as default method.

To declare default methods in interfaces we have to use "default" keyword in method syntax like access modifier.

Ex:

```
interface I{

default void m1(){

System.out.println("m1-A");

}}

class A implements I{}

class Test{

public static void main(String args[]){

I i=new A();

i.m1();

}}
```

NOTE: It is possible to provide more than one default methods with in a single interface.

Ex:

```
interface I{
default void m1(){
---------
}
default void m2(){
--------
}}
```

1.In JAVA8,it is possible to override default methods in the implementation classes.

```
interface I{
default void m1(){
System.out.println("m1-A");
}}
class A implements I{
public void m1(){
System.out.println("m1-A");
}}
class Test{
public static void main(String args[]){
I i=new A();
i.m1();
}}
```

2.Static Methods in Interfaces:

Upto JAVA7 version, static methods are not possible in interfaces but from JAVA8 version static methods are possible in interfaces in order to improve sharability.

If we declare static methods in the interfaces then it is not required to declare any implementation class to access that static method,we can use directly interface name to access static method.

NOTE:If we declare static methods in an interface then they will not be available to the respective implementation classes,we have to access static methods by using only interface names not even by using interface reference variable

```
interface I{

static void m1(){

System.out.println("m1-1");

}}

class Test{

public static void main(String args[]){

I.m1();

}}
```

Note:In JAVA8 version, interfaces will allow concrete methods along with either "static" keyword or "default" keyword.

3.Functional Interface:

If any Java interface allows only one abstract method then it is called as "Functional Interface".

To make any interface as Functional Interface then we have to use the following annotation just above of the interface.

@FunctionalInterface

Ex: java.lang.Runnable

java.lang.Comparable

NOTE: In Functional Interfaces we have to provide only one abstract method but we can provide any no.of default methods and any no.of static methods.


```
@Functional Interface
interface I{
void m1();
//void m2();---->error
default void m3(){
System.out.println("m3-I");
}
```

```
static void m4(){
System.out.println("m4-I");
}
class A implements I{
public void m1(){
System.out.println("m1-A");
}}
class Test{
public static void main(String args[]){
I i=new A();
i.m1();
i.m3();
//i.m4();--->error
I.m4();
//A.m4();--->error
}}
```

instanceof operator:

It is a boolean operator,it can be used to check whether the specified reference variable is representing the specified class object or not that is compatible or not.

ref_Var instanceof Class_Name

where ref_Var and Class_Name must be related otherwise compiler will rise an error like "incompatible types error".

If ref_Var class is same as the specified Class_Name then instanceof operator will return "true".

If ref_Var class is subclass to the specified Class_Name then instanceof operator will return "true".

If ref_Var class is super class to the specified Class_Name then instanceof operator will return "false".

```
class A{

}

class B extends A{

}

class C{

}

class Test{

public static void main(String args[]){

A a new A();

B b=new B();

System.out.println(a instanceof A);

System.out.println(a instanceof B);

System.out.println(b instanceof A);

//System.out.println(a instanceof C);

}}
```

Object Cloning:

The process of creating duplicate object for an existed object is called as Object Cloning.

If we want to perform Object Cloning in Java application then we have to use the following steps.

1.Declare an User defined Class.

2.Implement java.lang.Cloneable interface inorder to make eligible any object for cloning.

3.Override Object class clone() method in user defined class.

public Object clone() throws CloneNotSupportedException

4.In Main class,in main() method,access clone() method over the respective object.


```
class Student implements Cloneable{
String sid;
String sname;
String saddr;
Student(String sid,String sname,String saddr){
this.sid=sid;
this.sname=sname;
this.saddr=saddr;
}
public Object clone()throws CloneNotSupportedException{
```

```
return super.clone();
}
public String toString(){
System.out.println("Student details");
System.out.println("----------------");
System.out.println("Student Id :"+sid);
System.out.println("Student name:"+sname);
System.out.println("Student Address:"+saddr);
return "";
}}
class Test{
public static void main(String args[]){
Student std1=new Student("S-111","Durga","Hyd");
System.out.println("Student Details Before Cloning");
System.out.println(std1);
Student std2=(Student)std1.clone();
System.out.println();
System.out.println("Student Details After cloning");
System.out.println(std2);
}}
```

There are two types clonings in Java:

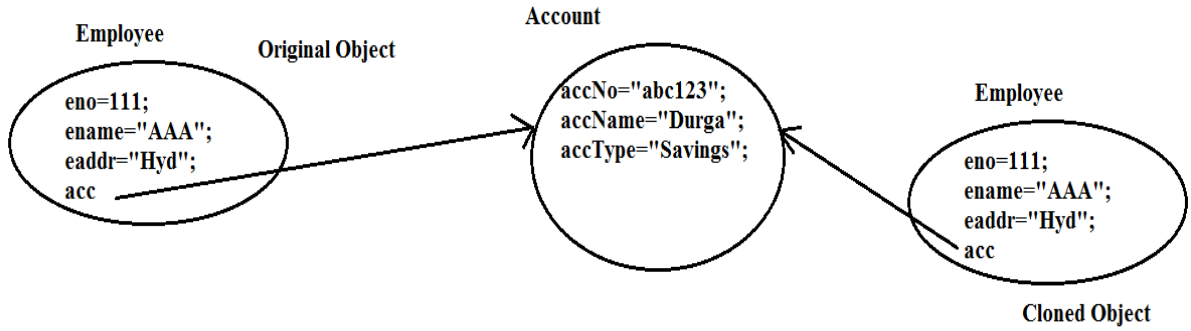
1.Shallow Cloning/Shallow Copy

2.Deep Cloning/Deep Copy

1.Shallow Cloning/Shallow Copy:

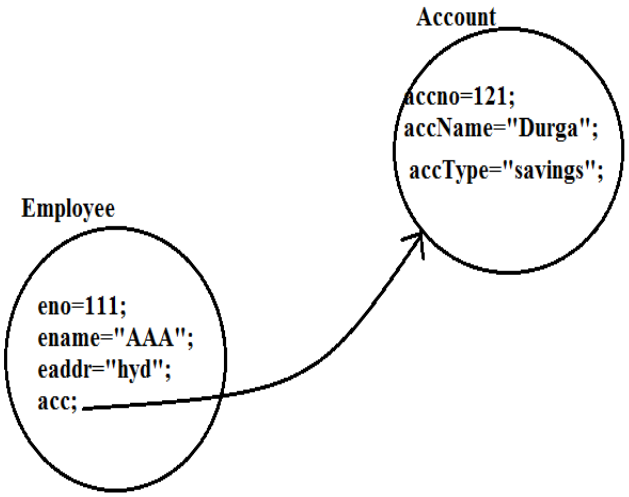
In this cloning mechanism,while cloning an object if any associated object is encountered then JVM will not duplicate associated object along with data duplication,where duplicated object is also refer the same associated object which was refereed by original object.

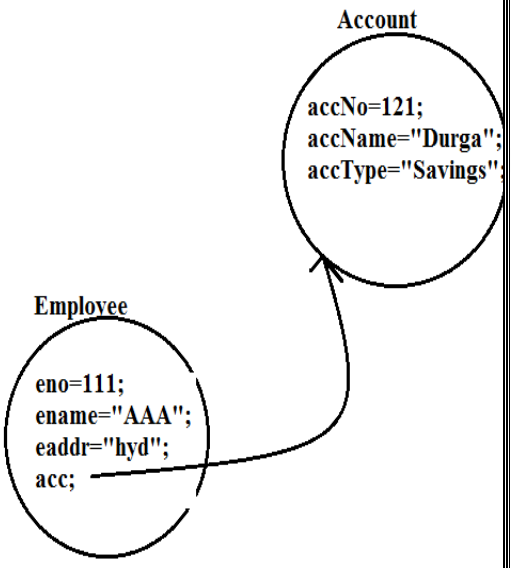
NOTE:Shallow Cloning is default cloning mechanism in Java.



Deep Cloning/Deep Copy:

In this cloning mechanism,while cloning an object if JVM encounter any associated object then JVM will duplicate associated object also along with data duplication.In this cloning mechanism,both original object and cloned object are having their own duplicated associated objects copy,both are not referring a single associated object






```
class Account{
String accNo;
String accName;
String accType;
Account(String accNo,String accName,String accType){
this.accNo=accNo;
this.accName=accName;
this.accType=accType;
}}
class Employee implements Cloneable{
String eid;
String ename;
String eddr;
```

```
Account acc;
Employee(String eid,String ename,String eaddr,Account acc){
this.eid=eid;
this.ename=ename;
this.eaddr=eaddr;
this.acc=acc;
}
public Object clone() throws CloneNotSuppoertException{
return super.clone();
}
public String toString(){
System.out.println("Employee Details");
System.out.println("-----------------");
System.out.println("Employee Id  :"+eid);
System.out.println("Employee Address :"+eaddr);
System.out.println("Employee Name  :"+ename);
System.out.println("Account Deatails");
System.out.println("---------------");
System.out.println("Account Number:"+acc.accNo);
System.out.println("Account Name :"+acc.accName);
System.out.println("Account Type :"+acc.accType);
System.out.println("Account Reference: "+acc);
return "";
}}
class Test{
public static void main(String args[]) throws Exception{
```

```
Account acc=new Account("abc123","Durga","Savings");

Employee emp1=new Employee("E-111","Durga","Hyd",acc);

System.out.println("Employee Details Before Cloning");

System.out.println(emp1);

Employee emp2=(Employee)emp1.clone();

System.out.println("Employee Details After Cloning");

System.out.println(emp2);

}}
```

To prepare example for deep cloning provide the following clone() method in employee class in the above example of(shallow Cloning)

```
public Object clone()throws CloneNotSupportedException{

Account acc1=new Account(acc.accNo,acc.accName,acc.accType);

Employee emp=new Employee(eid,ename,eaddr,acc1);

return emp;

}
```