Object Orientation

Object Orientation

To prepare applications we have to select a particular type of programming language from the following.

1) **Unstructured Programming Languages**
2) **Structured Programming Languages**
3) **Object Oriented Programming Languages**
4) **Aspect Oriented Programming Languages**

Q) What are the differences between Unstructured Programming Languages and Structured Programming Languages?

1) **Unstructured Programming Languages are out dated programming languages, they were introduced at starting point of computers and these programming languages are not suitable for our at present application requirements.**
   EX: BASIC, FOTRAN,.....

   Structured Programming languages are not out dated programming languages, these programming languages are utilized for our at present application requirements.
   EX: C , PASCAL,....

2) **Unstructured Programming Languages are not following any proper structure to prepare applications.**

   Structured Programming Languages are following a particular structure to prepare applications.

3) **Unstructured Programming languages are using mnemonic [ADD, SUB, MUL,..] [low level code] codes to prepare applications, which are available in very less number and which may provide very less no of features to prepare applications.**

   Structured Programming Languages are using high level syntaxes, which are available in more number and which may provide more no of features to the applications.

4) **Unstructured Programming Languages are using only "goto" statement to define flow of execution, which is not sufficient to provide very good flow of execution in applications.**

   Structured Programming Languages are using more and more no of flow controllers like if, switch, for, while, do-while, break,..... to defined very good flow of execution in applications.

5) Unstructured Programming languages are not having functions feature, it will increase code redundancy [Code duplication], it is not suggestible in application development.

   Structured Programming languages are having functions feature to improve code reusability.

Q) What are the differences between Structured Programming Languages and Object Oriented Programming Languages?

1) Structured Programming Languages are providing difficult approach to prepare applications.
   EX: C, PASCAL,...

   Object Oriented Programming Languages are providing very simplified approach to prepare applications.
   EX: JAVA, C++,....

2) Structured Programming languages are not having modularity.
   Object Oriented Programming Languages are having very good modularity.

3) Structured Programming languages are not having very good abstraction.
   Object Oriented Programming Languages are having very good abstraction levels.

4) Structured Programming languages are not providing very good security for the applications.
   Object oriented programming Languages is providing very good security for the application data.

5) Structured Programming languages are not providing very good sharability.
   Object oriented programming Languages are providing very good sharability.

6) Structured Programming Languages are not providing very good code reusability.
   Object oriented programming languages are providing very good code reusability.

Aspect Oriented Programming Languages:

If we want to prepare applications by using Object orientation then we have to provide both business logic and Services logic in combined manner, it will provide tightly coupled design, it will reduce sharability and code reusability.

To overcome the above problem we have to use Aspect Orientation. Aspect Orientation is a methodology or a set of rules and regulations or a set of guide lines which are applied on

object oriented programming to get loosely coupled design in order to improve sharability and code reusability.

In case of Aspect orientation, we will separate all services from business logic, we will declare each and every service as an aspect and we will inject these services to the applications at runtime as per the requirement.

Object Oriented Features:

To show 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

There are two types of programming languages on the basis of object oriented features.

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?

- Object Oriented Programming languages are able to allow all the object oriented features including "Inheritance".
  EX: Java

- Object based programming languages are able to allow all the object oriented features excluding "Inheritance".
  EX: Java Script

Q) What are the differences between class and Object?

1) Class is a group of elements having common properties and behaviors.
   Object is an individual element among the group of elements having physical behaviors and physical properties

2) Class is virtual.
   Object is real

3) Class is virtual encapsulation of properties and behaviors.
   Object is physical encapsulation of properties and behaviors.

4) Class is Generalization.
   Object is Specialization.

5) Class is Model or Blue Print for the objects.
   Object is an instance of the class.

Q) What is the difference between Encapsulation and Abstraction?

The process of binding data and coding part is called as "Encapsulation".

The process showing necessary data or implementation and hiding unnecessary data or implementation is called as "Abstraction".

Note: In Object Oriented programming languages, both encapsulation and abstraction are improvising "Security".

5.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".

EX:

```
1)  class Employee {
2)        String eid;
3)        String ename;
4)        String eaddr;
5)        ---
6)        ---
7)        void getEmpDetails() {
8)             ----
```

```
9)        }
10) }
11) class Manager extends Employee {
12)         --- Reuse variables and methods of Employee class---
13) }
14) class Accountant extends Employee {
15)          --- Reuse variables and methods of Employee class here----
16) }
17) class Engineer extends Employee {
18)         --- Reuse variables and methods of Employee class here----
19) }
```

6.Polymorphism:

- Polymorphism is "Greak" word, where poly means many and morphism means structures [forms].
- If one thing is existed in multiple forms then it is called as Polymorphism.
- The main advantage of Polymorphism is "Flexibility" to design application.

7.Message Passing:

The process of transferring data along with flow of execution from one instruction to another instructions is called as Message passing.

The main advantage of message passing is to improve "Communication" between entities and "data navigation" between entities.

Containers in Java:

Container is a Java element, it able to manage some other programming elements like variables, methods, blocks,.....

There are three types of containers in JAVA.

1) Class
2) Abstract Class
3) Interface

1) Class:

The main Purpose of the class is to represent all real world entities in Java applications.
EX: Student, Employee, Product, Account,....

To represent entities data in Java applications we will use Variables.
To represent entities behaviours we will use methods.

Syntax:

```
[Access_Modifiers] class Class_Name [extends Super_Class_Name]
        [implements interface_List]
        {
                --- variables-----
                -- methods--------
                ----constructors-----
                ----blocks------
                ---classes-------
                ----abstract classes-----
                ---interfaces------
                ---- enums--------
        }
```

Access Modifiers:

There are two types of Access modifiers

1.To define scopes to the programming elements, there are four types of access modifiers. public, protected, <default> and private.
Where public and <default> are allowed for classes, protected and private are not allowed for classes.

Note: All public, protected , <default> and private are allowed for inner classes.
Note: Where private and protected access modifiers are defined on the basis of classes boudaries, so that,they are not applicable for classes, they are applicable for members of the classes including inner classes.

2.To define some extra nature to the programming elements , we have the following access modifiers.

static, final, abstract, native, volatile, transient, synchronized, strictfp,.....

Where final, abstract and strictfp are allowed for the classes.

Note: The access modifiers like static, abstract, final and strictfp are allowed for inner classes.

Note: Where static keyword was defined on the basis of classes origin, so that, it is not applicable for classes, it is applicable for members of the classes including inner classes.

Where 'class' is a java keyword, it can be used to represent 'Class' object oriented feature.

Where 'Class_Name' is name is an identifier, it can be used to recognize the classes individually.

Where 'extends' keyword is able to specify a particular super class name in class syntax in order to get variables and methods from the specified super class to the present java class.

Note: In class syntax, 'extends' keyword is able to allow only one super class name, it will not allow more than one super class, because, it will represent multiple inheritance, it is not possible in Java.

Where 'implements' keyword is able to specify one or more no of interfaces in class syntax in order to provide implementation for all abstract methods of that interfaces in the present class.

Note: In class syntax, extends keyword is able to allow only one super class, but, implements keyword is able to allow more than one interface.

Note: In class syntax, 'extends' and 'implements' keywords are optional, we can write a class with extends keyword and without implements keyword, we can write a class with implements keyword and without extends keyword, we can write a class without both extends and implements keywords, if we want to write both extends and implements keywords first we have to write extends keyword then only we have to write implements keyword, we must not interchange extends and implements keywords.

1) class A{ } ----> valid
2) public class A{ }----> valid
3) private class A{ }----> Invalid
4) protected class A{ }----> Invalid
5) class A{ public class B{ } }----> valid
6) class A{ private class B{ } }----> valid
7) class A{ protected class B { } }----> valid
8) class A{ class B{ } }----> valid
9) static class A{ }----> Invalid
10) final class A{ }----> valid
11) abstract class A{ }----> valid
12) synchronized class A{ }----> Invalid

13) native class A{ }----> Invalid
14) class A{ static class B{ } }----> valid
15) class A{ abstract class B{ } }----> valid
16) class A{ strictfp class B{ } }----> valid
17) class A{ native class B{ } }----> Invalid
18) class A extends B{ }----> valid
19) class A extends A{ }----> Invalid, cyclic inheritance Error
20) class A extends B, C{ }----> Invalid
21) class A implements I{ }----> valid
22) class A implements I1, I2{ }----> valid
23) class A implements I extends B{ }----> Invalid
24) class A extends B implements I{ }----> valid
25) class A extends B implements I1, I2{ }----> valid

Procedure To Write Classes In Java Applications:

1) Declare a class by using 'class' keyword.
2) Declare variables and methods in class as per the requirement.
3) In main class and in main() method, create object for the respective class.
4) Access members of the class by using the generated reference variable.

Entities[Student, Customer, Employee,...]-----> classes
Entities data[sid, sname, saddr,...] ----> variables
Entities actions or behaviours[add, search, delete,..] ----> methods

EX:

```
1)  class Employee
2)  {
3)     String eid="E-111";
4)     String ename="Durga";
5)     float esal=25000.0f;
6)     String eaddr="Hyderabad";
7)     String eemail="durga@durgasoft.com";
8)  String emobile="91-9988776655";
9)     public void display_Emp_Details()  {
10)       System.out.println("Employee Details");
11)       System.out.println("-------------------");
12)       System.out.println("Employee Id     :"+eid);
13)       System.out.println("Employee Name   :"+ename);
14)       System.out.println("Employee Saslary:"+esal);
15)       System.out.println("Employee Address:"+eaddr);
16)       System.out.println("Employee Email  :"+eemail);
17)       System.out.println("Employee Mobile :"+emobile);
18)    }
19) }
```

```
20) class Test
21) {
22)    public static void main(String[] args)
23)    {
24)       Employee emp = new Employee();
25)       emp.display_Emp_Details();
26)    }
27) }
```

There are two types of methods.

1) Concrete methods
2) Abstract methods

Q) What are the differences between *Concrete methods* and *abstract methods*?

1) Concrete method is a method, it will have both method declaration and method implementation.

   EX: void add(int i, int j)//Method Declaration
   {// Method implementation started here
   int k=i+j;
   System.out.println(k);
   }// Method Implementation ended here

   Abstract method is a method, it will have only method declaration.

   EX: abstract void add(int i, int j);

2) To declare concrete methods, no need to use any special keyword.
   To declare abstract method, we must use abstract keyword.
3) Concrete methods are allowed in classes and in abstract classes.
   Abstract methods are allowed in abstract classes and interfaces.
4) Concrete methods will provide less sharability.
   Abstract methods will provide more sharability.

java 8 default and static methods are allowed in interface and java9 private methods

2) Abstract Classes:

Abstract class is a java class, it able to allow zero or more no of concrete methods and zero or more no of abstract methods.

Note: To declare abstract classes, it is not at all mandatory condition to have at least one abstract method, we can declare abstract classes with 0 no of abstract methods, but, if we want to declare a method as an abstract method then the respective class must be abstract class.

For abstract classes, we are able to create only reference variables; we are unable to create objects.

```
abstract class A {
      ----
}
A  a  =  new  A();--> Error
A a = null; → No Error
```

Abstract classes are able to provide more sharability when compared with classes.

Procedure to use Abstract Classes:

1) Declare an abstract class with "abstract".
2) Declare concrete methods and abstract methods in abstract class as per the requirement.
3) Declare sub class for abstract class.
4) Implement abstract methods in sub class.
5) In main class and in main() method, create object for sub class and declare reference variable either for abstract class or for sub class.
6) Access abstract class members.

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

EX:

```
1)  abstract class A
2)  {
3)     void m1()
4)     {
5)        System.out.println("m1-A");
6)     }
7)     abstract void m2();
8)     abstract void m3();
9)  }
10) class B extends A
11) {
12)    void m2()
13)    {
14)       System.out.println("m2-B");
15)    }
16)    void m3()
17)    {
18)       System.out.println("m3-B");
```

```
19)   }
20)   void m4()
21)   {
22)      System.out.println("m4-B");
23)   }
24) }
25) class Test
26) {
27)   public static void main(String[] args)
28)   {
29)      //A a=new A();--> Error
30)      A a = new B();
31)      a.m1();
32)      a.m2();
33)      a.m3();
34)      //a.m4();--> Error
35)      B b = new B();
36)      b.m1();
37)      b.m2();
38)      b.m3();
39)      b.m4();
40)   }
41) }
```

Q) What are the differences between *Concrete Classes* and *Abstract Classes?*

1) Classes are able to allow only concrete methods.
   Abstract classes are able to allow both concrete methods and abstract methods.

2) To declare concrete classes, only, 'class' keyword is sufficient.
   To declare abstract classes we need to use 'abstract' keyword along with class keyword.

3) For classes, 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.

4) Concrete classes will provide less sharability.
   Abstract classes will provide more sharability.

3) Interfaces:

- Interface is a java feature, it able to allow zero or more no of abstract methods only.
- For interfaces, we are able to declare only reference variables; we are unable to create objects.
- In case of interfaces, by default, all the variables as "public static final".
- In case of interfaces, by default, all the methods are "public and abstract".
- When compared with classes and abstract classes, interfaces will provide more sharability.

Procedure to Use interfaces in Java applications:

1) Declare an interface with "interface" keyword.
2) Declare variables and methods in interface as per the requirement.
3) Declare an implementation class for interface by including "implements" keyword.
4) Provide implementation for abstract methods in implementation class which are declared in interface.
5) In main class, in main() method, create object for implementation class ,but, declare reference variable either for interface or for implementation class.
6) Access interface members.

Note: If 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:

```
1)  interface I
2)  {
3)     int x=20;// public static final
4)     void m1();// public and abstract
5)     void m2();// public and abstract
6)     void m3();// public and abstract
7)  }
8)  class A implements I
9)  {
10)    public void m1()
11)    {
12)       System.out.println("m1-A");
13)    }
14)    public void m2()
15)    {
16)       System.out.println("m2-A");
17)    }
```

```
18)    public void m3()
19)    {
20)       System.out.println("m3-A");
21)    }
22)    public void m4()
23)    {
24)       System.out.println("m4-A");
25)    }
26) }
27) class Test
28) {
29)    public static void main(String[] args)
30)    {
31)       //I i=new I();---> Error
32)       I i=new A();
33)       System.out.println(I.x);
34)       System.out.println(i.x);
35)       i.m1();
36)       i.m2();
37)       i.m3();
38)       //i.m4();----> Error
39)       A a=new A();
40)       System.out.println(a.x);
41)       System.out.println(A.x);
42)       a.m1();
43)       a.m2();
44)       a.m3();
45)       a.m4();
46)    }
47) }
```

Q) What are the differences between Class, abstract Clacss and Interface?

1) Class is able to allow concrete methods only.
   Abstract class is able to allow both concrete methods and abstract methods
   Interface is able to allow abstract methods only.

2) To declare class, only, class keyword is sufficient.
   To declare abstract class, we have to use "abstract" keyword along with class keyword.
   To declare interface we have to use "interface" keyword explicitly.

3) For classes only, we are able to create both reference variables and objects.
   For abstract classes and interfaces, we are able to declare reference variables; we are unable to create objects.

4) In case of interfaces, by default, all variables are "public static final".
In case of classes and abstract classes, no default cases for variables.

5) In case of interfaces, by default, all methods are "public and abstract".
In case of classes and abstract classes, no default cases for methods.

6) Constructors are allowed in classes and abstract classes.
Constructors are not allowed in interfaces.

7) Classes are able to provide less sharability.
Abstract classes are able to provide moddle level sharability.
Interfaces are able to provide more sharability.

classes are allowed constructors, static, instance block also abstract class also allowed
interfaces are not allowed constructors, static, instance block

Methods In Java:

Method is a set of instructions, it will represent a particular action in java applications.

Syntax:

```
[Access_Modifiers] return_Type method_Name([param_List])[throws Exception_List]
{
  ----- instructions to represent a particular action-----
}
```

Java methods are able to allow the access modifiers like public, protected, <default> and private.
Java methods are able to allow the access modifiers like static, final, abstract, native, synchronized, strictfp.

Note - java methods are able to allow more than one access modifier , but they must be provided in valid combination

Where the purpose of return_Type is to specify which type of data the present method is returning. In java applications, all primitive data types, all user defined data types and 'void' are allowed for methods as return types.

Note: 'void' return type is representing "Nothing is returned" from methods.

Where "method_name" is an identifier, it can be used to recognize the methods individually.

Where param_List can be used to pass some input data to the methods in order to perform an action. In java applications, we are able to provide all primitive data types and all user defined data types as parameter types.

Where "throws" keyword can be used to bypass or delegate the generated exception from present method to caller method of the present method

To describe method information, there are two approaches.

1) Method Signature
2) Method Prototype

Q) What is the difference between *Method Signature* and *Method Prototype?*

Method signature is the description of the method, it includes method name and parameter list.

EX: forName(String Class_Name)

Method prototype is the description of the method, it will include access modifiers list, return type, method name, parameters list, throws exception list.

EX:public static Class forName(String class_Name)throws ClassNotFoundException

There are two types of methods in Java w.r.t the Object state manipulations.

1) Mutator Methods
2) Accessor Methods

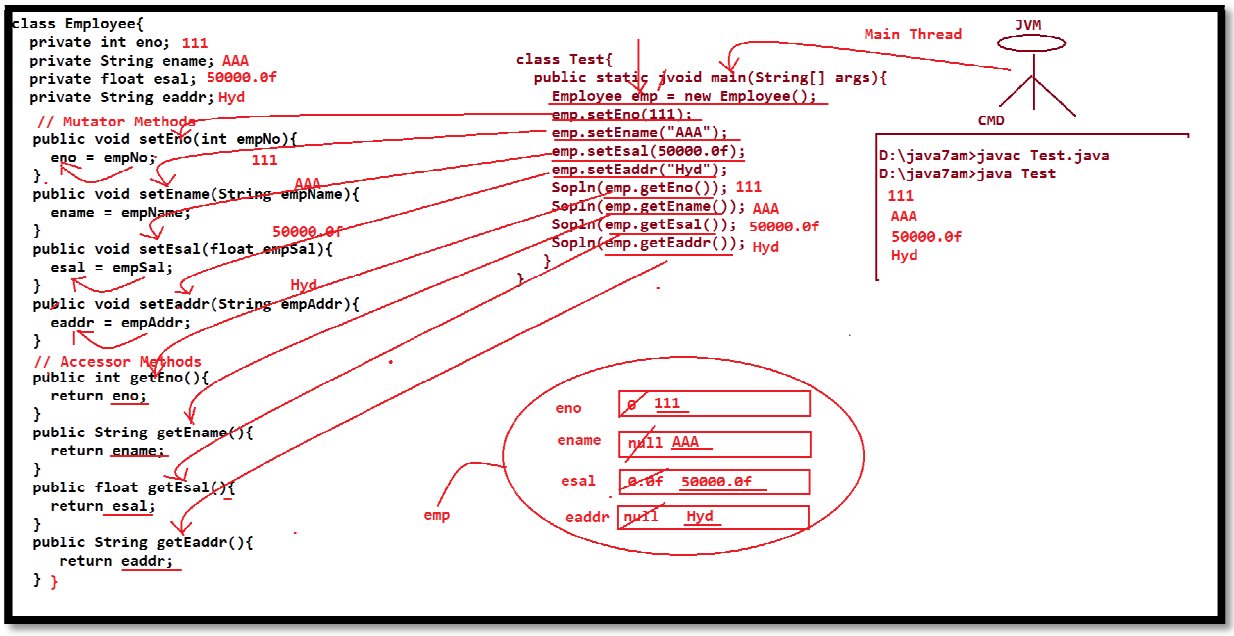
Q) What are the differences between *Mutator methods* and *Accessor methods?*

Mutator methods are the java methods, which are used to set/modify data in Objects.

EX: All setXXX(--) methods in Java Bean classes are mutator methods.

Accessor methods are the java methods, which are used to get/access data from Objects.

EX:All getXXX() methods in java bean classes are Accessor methods



Note: Java bean is a normal java class, it will include properties and the respective setXXX(-) methods and getXXX() methods.

Variable-Argument Method [Var-Arg method]

In general, in java applications, if we declare any method with 'n' no of parameters then we must access that method with the same 'n' no of parameter values , it is not possible to access that method by passing 'n+1' no of parameter values and 'n-1' no of parameter values.

EX:

```
1)  class A
2)  {
3)     void add(int i, int j)
4)     {
5)        System.out.println(i+j);
6)     }
7)  }
8)  class Test
9)  {
10)    public static void main(String[] args)
11)    {
12)       A a=new A();
13)       a.add(10,20);// Valid.
14)       //a.add();--> Invalid
15)       //a.add(10);--> Invalid
16)       //a.add(10,20,30);--> Invalid
17)    }
18) }
```

As per the requirement, we want to access a method with variable no of parameter values [0 no of params or 1 no of params or 2 no of params,....]

To achieve this requirement we have to use "Var-Arg Method" provided by JDK5.0 version.

Var-Arg method is a java method including Var-Arg parameter.
Syntax for Var-Arg Parameter:

Data_Type ... Var_Name

EX: for var-Arg method:

```
void m1(int ... a)//int[] a={---- argumentvalues-----}
{
  -----
}
```

When we access Var-Arg method with 'n' no of parameter values then all these 'n' no of parameter values are stored in the form of Array which is generated from var-ARg parameter.

EX:

```
1)  class A
2)  {
3)     void add(int ... a)// int[] a={--- listof arguments---}
4)     {
5)        System.out.println("No Of Arguments  :"+a.length);
6)        int result=0;
7)        System.out.print("Argument List    :");
8)        for(int i=0;i<a.length;i++)
9)        {
10)          System.out.print(a[i]+"  ");
11)          result=result+a[i];
12)       }
13)       System.out.println();
14)       System.out.println("Addition        :"+result);
15)       System.out.println("--------------------------");
16)    }
17) }
18) class Test
19) {
20)    public static void main(String[] args)
21)    {
22)       A a=new A();
23)       a.add();
24)       a.add(10);
25)       a.add(10,20);
26)       a.add(10,20,30);
27)    }
28) }
```

Note: In Var-Arg methods, normal parameters are allowed along with Var-Arg parameter but they must be provided before var-Arg parameter, not after var-Arg parameter, because, var-Arg parameter must be last parameter in var-Arg method.

Note: Due to the above reason, Var-Arg methods are able to allow at most one Var-Arg parameter, not allowing more than one Var-Arg parameters.

EX:

1) void m1(int ... i, float f){ } ----> Invalid
2) void m1(float f, int ... i){ } ----> valid
3) void m1(int ... i, float ... f){ } --->Invalid

Object Creation Process:

Q)What is the Requirement to Create Objects in Java?

1) To store entities data temporarily in java applications we need objects.
2) To Access the members of any particular class we need objects.

If we want to create Objects we have to use the following syntax.
Class_Name ref_Var = new Class_Name([param_List]);

EX:

```
class A{
----
}

A  a = new  A();
```

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

1) Creating memory for the Object
2) Generating Identities for the object
3) Providing initializations inside the Object

1) Creating Memory for the object:

When JVM encounter "new" keyword in Object creation statement then JVM will check to which class we are creating object on the basis of Constructor name then JVM will load the respective class byte code to the memory [Method Area]

After loading class byte code, JVM will identify minimal object memory size by recognizing all the instance variables and their data types

After getting memory size, JVM will send a request to Heap manager about to create an object with the specified minimal memory.

As per JVM requirement, Heap Manager will create the required block of memory at Heap Memory.

2) Generating Identities for the Object:

After creating a block of memory [Object] for JVM requirements, Heap manager will assign an integer value as an identity for the object called as "HashCode".

After getting HashCode value, Heap Manager will send that hashCode value to JVM, where JVM will convert that hashCode value to its Hexa Decimal form called as "Reference Value".

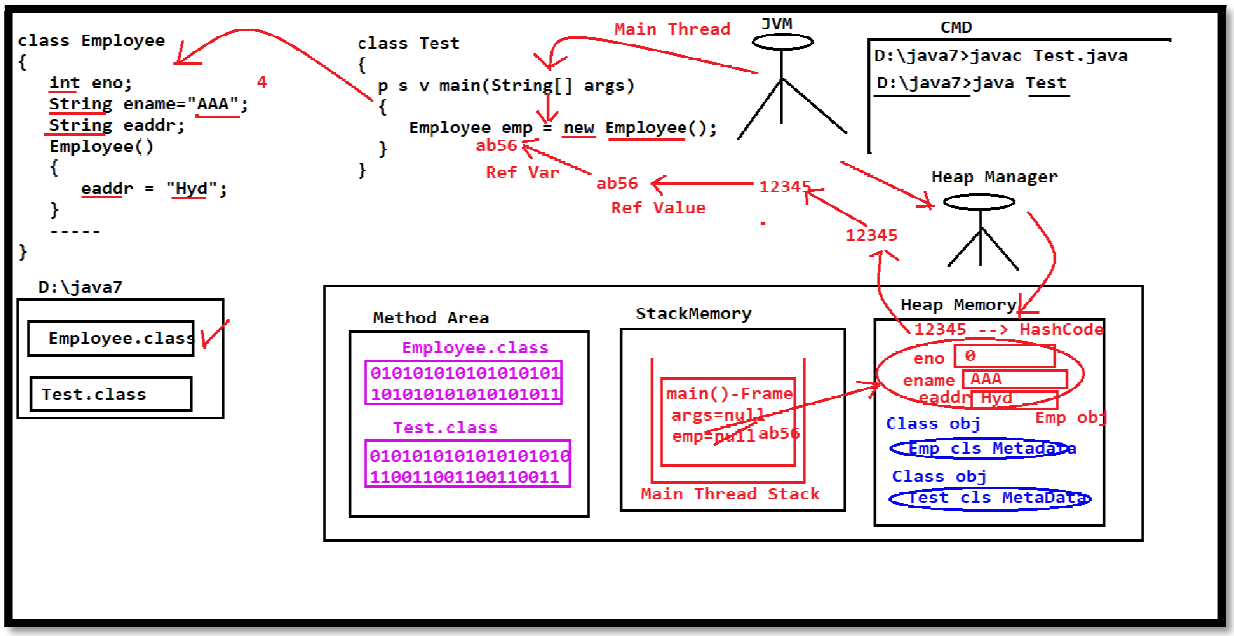
After getting Object reference value, JVM will assign that reference value to a variable called as "Reference Variable".

3) Providing initializations inside the Object:

After creating object and its identities, JVM will allocate memory for all the instance variables inside the object on the basis of their data types.

After getting memory for the instance variables, JVM will provide initial values to the instance variables by searching initializations at class level declaration and at constructor.

If any instance variable is not having initialization at both constructor and at class level declaration then JVM will store default value on the basis of their data type as initial value inside the object.



Note: In java applications, when a class byte code is loaded to the memory, automatically, JVM will create java.lang.Class object at heap memory with the metadata of the loaded class.

Note: In java application, when we create a thread [Main Thread or User Thread], automatically, a stack will be created at stack memory and it able to store methods details in the form of Frames which are accessed by the respective thread.

To get hashcode value and reference value of an object we have to use the following two methods.

public native int hashCode()
public String toString()

Note: Native method is a java method, declared in Java, but, implemented in non Java programming languages may be C, ASL, ...

EX:

```
1)  class A
2)  {
3)  }
4)  class Test
5)  {
6)     public static void main(String[] args)
7)     {
8)        A a=new A();
9)        int hashCode=a.hashCode();
10)       System.out.println("Object HashCode  :"+hashCode);
11)       String ref=a.toString();
12)       System.out.println("Object Reference :"+ref);
13)    }
14) }
```

OUTPUT:
Object Hashcode: 31168322
Object Reference: A@adb9742

In Java applications, there is a common and default super class for each and every java class [predefined classes and user defined classes] that is "java.lang.Object" class, where java.lang.Object class contains the following 11 methods in order to share to all java classes.

1) hashCode()
2) toString()
3) getClass()
4) clone()
5) equals(Object obj)
6) finalize()
7) wait()

8) wait(long time)
9) wait(long time, int time)
10) notify()
11) notifyAll()

Q) If we extend a class to some other class explicitly then the respective sub class is having two super classes [default super class [Object] and explicit super class], it represents multiple inheritance, how can we say multiple inheritance is not possible in JAVA?

In java, java.lang.Object class is common and default super class for every class when that class is not extending from any other super class explicitly. If our class is extending some other class explicitly then Object class is not directly super class to the respective sub class, Object class is indirectly super class to the respective sub class through the explicit super class , that is, Object class is not super class to the respective sub class through Multiple Inheritance, that is through Multi Level Inheritance.

EX:

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

Note: Here Object class is super class to A, A is super class to B
Object <--- A <--- B

In java applications, when we pass a particular class object reference variable as parameter to System.out.println(-) method then JVM will access toString() over the provided reference variable internally.

EX:

```
1)  class A
2)  {
3)  }
4)  class Test
5)  {
6)      public static void main(String[] args)
7)      {
8)          A a=new A();
9)
10)         String ref=a.toString();
```

```
11)       System.out.println(ref);
12)
13)       System.out.println(a.toString());
14)
15)       System.out.println(a);//System.out.println(a.toString());
16)    }
17) }
```

OUTPUT:

```
A@1db9742
A@1db9742
A@1db9742
```

In the above program, when we access toString() method internally or externally , first, JVM has to execute toString(), to execute toString() method JVM will search for toString() method in the respective class whose reference variable we passed as parameter to System.out.println(--) method, if the required toString() method is not existed in the respective class then JVM will search for toString() method in its super class, here if super class is not existed then JVM will search for toString() method in the common and default super class Object class. In Object class, toString() method was implemented insuch a way to return a string contains "Class_Name@ref_Val" .

As per the requirement, if we want to display our own data instead of Object reference value when we pass reference variable as parameter to System.out.println(-) method then we have to provide our own toSrtring() method in the respective class.

EX:

```
1)  class Employee
2)  {
3)     String eid="E-111";
4)     String ename="Durga";
5)     float esal=50000.0f;
6)     String eaddr="Hyd";
7)     String eemail="durga@durgasoft.com";
8)     String emobile="91-9988776655";
9)
10)    public String toString()
11)    {
12)       System.out.println("Employee Details");
13)       System.out.println("----------------------");
14)       System.out.println("Employee Id     :"+eid);
15)       System.out.println("Employee Name   :"+ename);
16)       System.out.println("Employee Salary :"+esal);
17)       System.out.println("Employee Address :"+eaddr);
18)       System.out.println("Employee Email  :"+eemail);
```

```
19)       System.out.println("Employee Mobile  :"+emobile);
20)       return "";
21)    }
22) }
23) class Test
24) {
25)    public static void main(String[] args)
26)    {
27)       Employee emp=new Employee();
28)       System.out.println(emp);
29)    }
30) }
```

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

EX:

```
1)  class Test
2)  {
3)     public static void main(String[] args)
4)     {
5)        String str=new String("abc");
6)        System.out.println(str);
7)
8)        Exception e=new Exception("My Own Exception");
9)        System.out.println(e);
10)
11)       Thread t=new Thread();
12)       System.out.println(t);
13)
14)       Integer in=new Integer(10);
15)       System.out.println(in);
16)
17)       java.util.ArrayList al=new java.util.ArrayList();
18)       al.add("AAA");
19)       al.add("BBB");
20)       al.add("CCC");
21)       al.add("DDD");
22)       System.out.println(al);
23)    }
24) }
```

OUTPUT:

```
abc
java.lang.Exception: My Own Exception
Thread[Thread-0,5,main]
10
[AAA,BBB,CCC,DDD]
```

In JAVA, there are two types of Objects.

1) 1.Immutable Objects
2) 2.Mutable Objects

Q) What is the difference between *Immutable object* and *mutable object?*

OR

Q) What is the difference between *String* and *StringBuffer?*

Immutable Objects are Java objects; they will not allow modifications on their content. If we are trying to perform operations over immutable objects content then data is allowed for operations, but, the resultant data is not stored back in original object, the modified data will be stored by creating new Object.

EX: All String class objects are immutable objects.
All Wrapper class objects are immutable Objects.

Mutable Objects are java objects, they will allow modifications on their content directly.
EX: Bydefault, all JAVA objects are mutable objects.
EX: StringBuffer

EX:

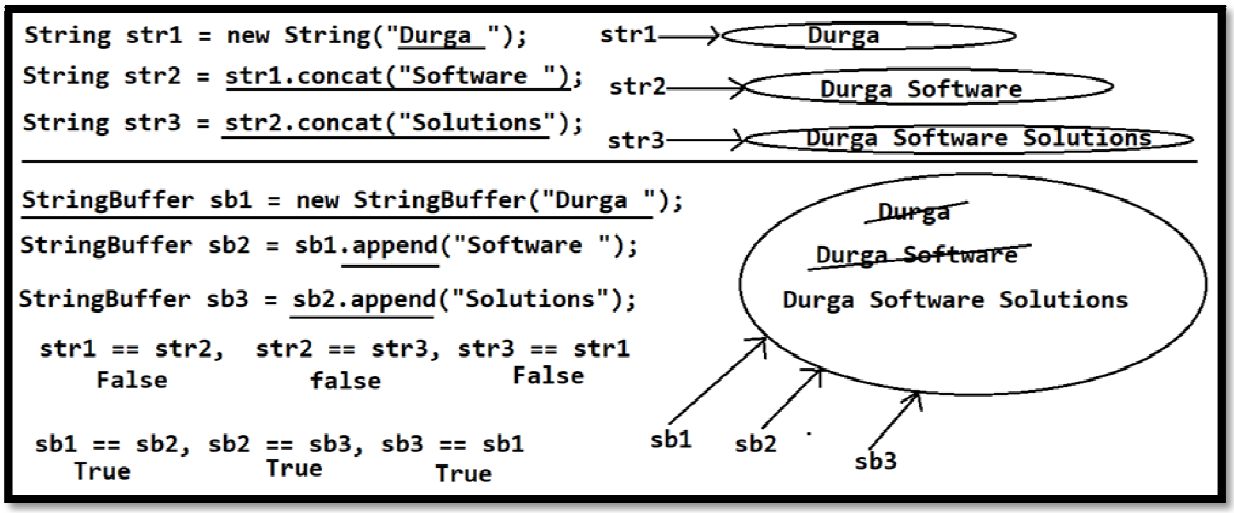
```
1)  class Test
2)  {
3)     public static void main(String[] args)
4)     {
5)        String str1=new String("Durga ");
6)        String str2=str1.concat("Software ");
7)        String str3=str2.concat("Solutions");
8)        System.out.println(str1);
9)        System.out.println(str2);
10)       System.out.println(str3);
11)       System.out.println();
12)       StringBuffer sb1=new StringBuffer("Durga ");
13)       StringBuffer sb2=sb1.append("Software ");
14)       StringBuffer sb3=sb2.append("Solutions");
```

```
15)      System.out.println(sb1);
16)      System.out.println(sb2);
17)      System.out.println(sb3);
18)   }
19)}
```

OUTPUT:

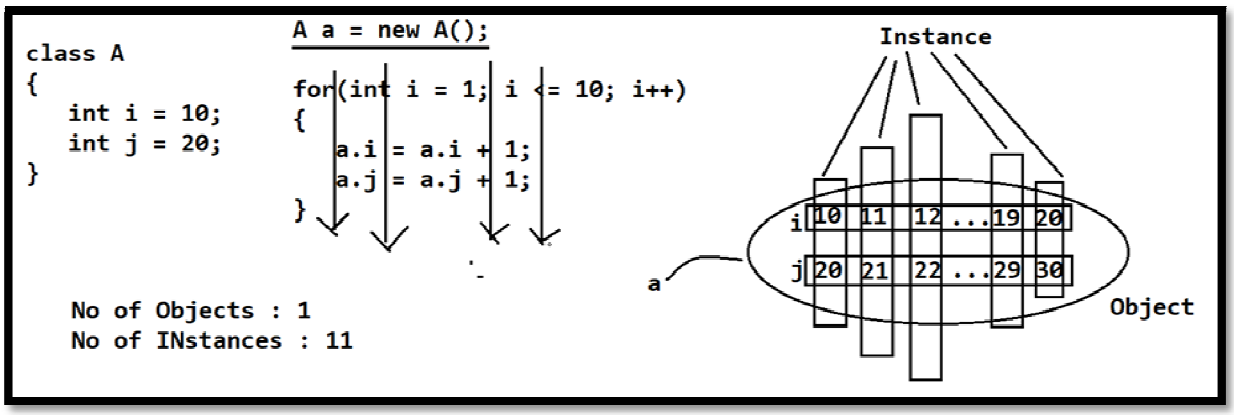
Durga
Durga Software
Durga Software Solutions

Durga Software Solutions
Durga Software Solutions
Durga Software Solutions



Q) What is the difference between Object and instance?

Object is a memory unit to store data.
Instance is a copy or layer of values existed in an object at a particular point of time.



Constructors:

1) Constructor is a java feature; it can be used to create Object.
2) The role of the constructors in Object creation is to provide initial values inside the object.
3) In java applications, Constructors are executed exactly at the time of creating objects, not before creating objects and not after creating objects.
4) The main utilization of constructors is to provide initializations for class level variables mainly instance variable.
5) Constructors must have same name of the respective class.
6) Constructors are not having return types.
7) Constructors are not allowing the access modifiers like static, final,...
8) Constructors are able to allow the access modifiers like public, protected,<default> and private.
9) Constructors are allowing 'throws' keyword in its syntax to bypass exceptions from present constructor to the caller.

Syntax: [Access_Modifier] Class_Name([Param_List])[throws Exception_List]
{

}

Note 1:

If we provide constructor name other than class name then compiler will rise an error like "Invalid Method declaration, return type required", because, compiler has treated the provided constructor as normal java method without the return type, but, for methods return is mandatory.

EX:

```
1)  class A
2)  {
3)     B()
4)     {
5)        System.out.println("A-Con");
6)     }
7)  }
8)  class Test
9)  {
10)    public static void main(String[] args)
11)    {
12)       A a=new A();
13)    }
14) }
```

Status: Compilation Error

Note 2:

If we provide return type to the constructors then Compiler will not rise any error and JVM will not rise any exception and JVM will not provide any output, because, the provided constructor is converted as normal java method. In this context, if we access the provided constructor as normal java method then it will be executed as like normal java method.

EX:

```
1)  class A
2)  {
3)     void A()
4)     {
5)        System.out.println("A-Con");
6)     }
7)  }
8)  class Test
9)  {
10)    public static void main(String[] args)
11)    {
12)       A a=new A();
13)       a.A();
14)    }
15) }
```

Status: No Compialtion Error
OUTPUT: A-Con

Note 3:

If we provide the access modifiers like static, final,... to the constructors then Compiler will rise an error like "modifier xxx not allowed here".

EX:

```
1)  class A
2)  {
3)     static A()
4)     {
5)        System.out.println("A-Con");
6)     }
7)  }
8)  class Test
9)  {
```

```
10)   public static void main(String[] args)
11)   {
12)      A a=new A();
13)
14)   }
15) }
```

Status: Compilation Error.

Note 4:

If we declare constructor as "private" then that constructor is available upto the respective class only, not available to out side of the respective class.
In this context, If we want to create object for the respective class then it is possible inside the same class only.

EX:

```
1)  class A
2)  {
3)     private A()
4)     {
5)        System.out.println("A-Con");
6)     }
7)  }
8)  class Test
9)  {
10)   public static void main(String[] args)
11)   {
12)      A a=new A();
13)   }
14) }
```

Status: Compilation Error

There are two types constructors in java
1) Default Constructors
2) User Defined Constructors

1) Default Constructors:

If we have not provided any constructor explicitly in java class then compiler will provide a 0-arg constructor automatically, here the compiler provided 0-arg constructor is called as "Default Constructor".

If we provide any constructor explicitly then compiler will not provide any default constructor.

D:\javaapps\ Test.java

```
public class Test {
}
```

On Command Prompt:

```
D:\javaapps>javac Test.java
D:\javaapps>javap Test
Compiled from Test.java
public class Test {
  public Test() { -----> Default constructor
  }
}
```

Note: Default Constructors are having the same scope of the respective class.

2) User Defined Constructors:

- These constructors are provided by the developers as per their application requirements.
- If we provide any constructor explicitly without parameters then that constructor is called as 0-arg constructor.
- If we provide any constructor with at least one parameter then that constructor is called as parameterized Constructor.

Note: By default, all the default constructors are 0-arg constructors, but, all 0-arg constructors are not default constructors, some 0-arg constructors are provided by the compiler are called as Default Constructors and some other 0-arg constructors are provided by the users they are called as User defined constructors.

EX:

```
1)  class Employee
2)  {
3)     String eid;
4)     String ename;
5)     float esal;
6)     String eaddr;
7)
8)     Employee()
9)     {
10)       eid="E-111";
11)       ename="Durga";
12)       esal=50000.0f;
13)       eaddr="Hyd";
14)    }
```

```
15)
16)    public void getEmpDetails()
17)    {
18)       System.out.println("Employee Details");
19)       System.out.println("----------------------");
20)       System.out.println("Employee Id      :"+eid);
21)       System.out.println("Employee Name    :"+ename);
22)       System.out.println("Employee Salary  :"+esal);
23)       System.out.println("Employee Address :"+eaddr);
24)    }
25) }
26) class Test
27) {
28)    public static void main(String[] args)
29)    {
30)       Employee emp=new Employee();
31)       emp.getEmpDetails();
32)    }
33) }
```

In the above program, if we create more than one object for the Employee class by executing 0-arg constructor then same data will be stored in all multiple objects of the Employee.

As per the requirement, if we want to create multiple Employee objects with different data then we have to provide data to the Objects explicitly, for this we have to use parameterized constructor.

EX:

```
1)  class Employee
2)  {
3)     String eid;
4)     String ename;
5)     float esal;
6)     String eaddr;
7)
8)     Employee(String emp_Id, String emp_Name, float emp_Sal, String emp_Addr)
9)     {
10)       eid=emp_Id;
11)       ename=emp_Name;
12)       esal=emp_Sal;
13)       eaddr=emp_Addr;
14)    }
15)
16)    public void getEmpDetails()
```

```
17)   {
18)      System.out.println("Employee Details");
19)      System.out.println("----------------------");
20)      System.out.println("Employee Id      :"+eid);
21)      System.out.println("Employee Name    :"+ename);
22)      System.out.println("Employee Salary  :"+esal);
23)      System.out.println("Employee Address :"+eaddr);
24)   }
25) }
26) class Test
27) {
28)   public static void main(String[] args)
29)   {
30)      Employee emp1=new Employee("E-111", "Durga", 50000.0f, "Hyd");
31)      emp1.getEmpDetails();
32)      System.out.println();
33)      Employee emp2=new Employee("E-222", "Anil", 60000.0f, "Hyd");
34)      emp2.getEmpDetails();
35)      System.out.println();
36)      Employee emp3=new Employee("E-333", "Rahul", 80000.0f, "Hyd");
37)      emp3.getEmpDetails();
38)   }
39) }
```

Constructor Overloading:

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

EX:

```
1)  class A
2)  {
3)     int i,j,k;
4)     A()
5)     {
6)     }
7)     A(int i1)
8)     {
9)        i=i1;
10)    }
11)    A(int i1, int j1)
12)    {
13)       i=i1;
14)       j=j1;
15)    }
```

```
16)   A(int i1, int j1, int k1)
17)   {
18)      i=i1;
19)      j=j1;
20)      k=k1;
21)   }
22)   void add()
23)   {
24)      System.out.println("Addition  :"+(i+j+k));
25)   }
26) }
27) class Test
28) {
29)   public static void main(String[] args)
30)   {
31)      A a1=new A();
32)      a1.add();
33)      A a2=new A(10);
34)      a2.add();
35)      A a3=new A(10,20);
36)      a3.add();
37)      A a4=new A(10,20,30);
38)      a4.add();
39)   }
40) }
```

Note: If we declare more than one method with the same name and with different parameter list then it is called as "Method Overloading"

Instance Context/Instance Flow of Execution:

In Java, for every class loading a separate context will be created called as Static Context and for every Object a separate context will be created called as Instance context.

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

1) Instance Variables
2) Instance Methods
3) Instance Blocks

1) Instance Variables:

- Instance Variable 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 in "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 won't be executed without the method call.

EX:

```
1)  class A {
2)      int i=m1();
3)      A() {
4)       System.out.println("A-Con");
5)      }
6)      int m1() {
7)        System.out.println("M1-A");
8)        return 10;
9)      }
10) }
11) class Test {
12)    public static void main(String args[]) {
13)       A a = new A();
14)    }
15) }
```

OUTPUT:

M1-A
A-con

EX:

```
1)  class A {
2)      int j = m1();
3)      int m2() {
4)          System.out.println("m2-A");
5)          return 10;
6)      }
7)      A() {
8)              System.out.println("A-con");
9)      }
10)     int m1() {
11)                 System.out.println("m1-A");
12)                 return 20;
13)     }
14)     int i = m2();
15) }
16) class Test {
17)       public static void main(String args[]) {
18)               A a = new A();
19)       }
20) }
```

OUTPUT: m1-A
m2-A
A-con

3) 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 as like constructors.

Syntax:

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

EX:

```
1)  class A {
2)      A() {
3)              System.out.println("A-CON");
4)      }
5)      {
6)          System.out.println("IB-A");
7)      }
```

```
8)  }
9)  class Test {
10)     public static void main(String args[]) {
11)         A a = new A();
12)     }
13) }
```

OUTPUT:

IB-A
A-CON

EX:

```
1)  class A {
2)      A() {
3)          System.out.println("A-CON");
4)      }
5)      {
6)          System.out.println("IB-A");
7)      }
8)      int m1() {
9)          System.out.println("m1-A");
10)         return 10;
11)     }
12)     int i = m1();
13) }
14) class Test {
15)     public static void main(String args[]) {
16)         A a = new A();
17)     }
18) }
```

OUTPUT: IB-A
m1-A
A-CON

EX:

```
1)  class A {
2)      int m1() {
3)          System.out.println("m1-A");
4)          return 10;
5)      }
6)      {
7)          System.out.println("IB-A");
8)      }
```

```
9)      int i=m2();
10)     A() {
11)          System.out.println("A-con");
12)     }
13)     int i=m1();
14)     {
15)          System.out.println("IB1-A");
16)     }
17)     int m2();
18)     {
19)        System.out.println("m2-A");
20)        return 20;
21)     }
22) }
23) class Test {
24)      public static void main(String args[]) {
25)         A a1=new A();
26)         A a2=new A();
27)      }
28) }
```

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 4 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 provide 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 then JVM will search for that variables at class level, even at class level also if 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.

EX:

```
1)  class A {
2)      int i=10;
3)      int j=20;
4)      A(int i,int j) {
5)          System.out.println(i+" "+j);
6)          System.out.println(this.i+" "+this.j);
7)      }
8)  }
9)  class Test {
10)     public static void main(String args[]) {
11)         A a = new A(30,40);
12)     }
13) }
```

OUTPUT:

30 40
10 20

NOTE: 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, we have to declare parameter variable with the same name of the respective class level variable, here we have to transfer data from local variable to class level variable, for this, we have to assign local variable to class level variable. In this context, to refer class level variable over local variable separately we have to use 'this' keyword. In getter methods, always, we have to return class level variables only, so that, it is not required to use 'this' keyword.

EX:

```
1)  class User {
2)        private String uname;
3)        private String upwd;
4)        public void setUname(String uname) {
5)              this.uname = uname;
6)        }
7)        public void setUpwd(String upwd) {
8)              this.upwd = upwd;
9)        }
10)       public getUname() {
11)          return uname;
12)       }
13)       public getUpwd() {
14)       }
15) }
16) class Test {
17)       public static void main(String[] args) {
18)             User u = new User();
19)             u.setUname("abc");
20)             u.setUpwd("abc123");
21)             System.out.println("User Login Details");
22)             System.out.println("---------------------");
23)             System.out.println("User Name   :"+u.getUname());
24)             System.out.println("Password    :"+getEaddr());
25)       }
26) }
```

EX:

```
1)  class Employee
2)  {
3)     private String eid;
4)     private String ename;
5)     private float esal;
6)     private String eaddr;
7)
8)     public void setEid(String eid)
9)     {
10)       this.eid=eid;
11)    }
12)    public void setEname(String ename)
13)    {
14)       this.ename=ename;
15)    }
16)    public void setEsal(float esal)
17)    {
```

```
18)     this.esal=esal;
19)   }
20)   public void setEaddr(String eaddr)
21)   {
22)     this.eaddr=eaddr;
23)   }
24)
25)   public String getEid()
26)   {
27)     return eid;
28)   }
29)   public String getEname()
30)   {
31)     return ename;
32)   }
33)   public float getEsal()
34)   {
35)     return esal;
36)   }
37)   public String getEaddr()
38)   {
39)     return eaddr;
40)   }
41) }
42) class Test
43) {
44)   public static void main(String[] args)
45)   {
46)     Employee emp=new Employee();
47)     emp.setEid("E-111");
48)     emp.setEname("AAA");
49)     emp.setEsal(5000.0f);
50)     emp.setEaddr("Hyd");
51)
52)     System.out.println("Employee Details");
53)     System.out.println("--------------------");
54)     System.out.println("Employee Id     :"+emp.getEid());
55)     System.out.println("Employee Name   :"+emp.getEname());
56)     System.out.println("Employee Salary :"+emp.getEsal());
57)     System.out.println("Employee Address :"+emp.getEaddr());
58)   }
59) }
```

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]);

Note: To access current class methods, it is not required to use any reference variable and any keyword including 'this', directly we can access.

EX:

```
1)  class A {
2)      void m1() {
3)          System.out.println("m1-A");
4)      }
5)      void m2() {
6)          System.out.println("m2-A");
7)          m1();
8)          this.m1();
9)      }
10) }
11) class Test {
12)     public static void main(String args[]) {
13)         A a = new A();
14)         a.m2();
15)     }
16) }
```

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]);

EX:

```
1)  class A {
2)      A() {
3)              this(10);
4)              System.out.println("A-0-arg-con");
5)      }
```

```
6)      A(int i) {
7)                  this(22.22f);
8)                  System.out.println("A-int-param-con");
9)      }
10)     A(float f) {
11)                 this(33.3333);
12)                 System.out.println("A-float-param-con");
13)     }
14)     A(double d) {
15)             System.out.println("A-double-param-con");
16)     }
17) }
18) class Test {
19)      public static void main(String args[]) {
20)           A a = new A();
21)      }
22) }
```

OUTPUT:

```
A-double-param-con
A-float-param-con
A-int-param-con
A-0-arg-con
```

NOTE: In the above program 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 program, we have called all the current class constructors by using 'this' keyword in chain fashion, this process 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, if we have not provided 'this' statement as first statement then compiler will rise an error like 'call to this must be first statement in constructor'.

EX:

```
1)  class A {
2)      A() {
3)              System.out.println("A-0-arg-con");
4)              this(10);
5)      }
6)      A(int i) {
7)                  System.out.println("A-int-param-con");
8)                  this(22.22f);
```

```
9)      }
10)     A(float f) {
11)                     System.out.println("A-float-param-con");
12)                     this(33.3333);
13)     }
14)     A(double d) {
15)                       System.out.println("A-double-param-con");
16)     }
17) }
18) class Test {
19)       public static void main(String args[]) {
20)               A a = new A();
21)       }
22) }
```

Status: Compilation Error

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 only, not in normal Java methods. If we violate this condition then compiler will rise an error like 'call to this must be first statement in constructors'.

EX:

```
1)  class A
2)  {
3)     A()
4)     {
5)        System.out.println("A-0-arg-con");
6)     }
7)     A(int i)
8)     {
9)        System.out.println("A-int-param-con");
10)    }
11)    void m1()
12)    {
13)       this(10);
14)       System.out.println("m1-A");
15)    }
16) }
17) class Test
18) {
19)    public static void main(String args[])
20)    {
21)       A a=new A();
```

```
22)      a.m1();
23)   }
24) }
```

Status: Compilation Error.

Q) Is it possible to refer more than one current class constructors by using 'this' keyword from a single current class constructor?

No, It is not possible to refer more than one current class constructors by using 'this' keyword, because, in constructors, 'this' statement must be first statement while referring current class constructors. If we provide more than one time this(---) statement then only one this() statement is first statement among multiple this statements, in this case, compiler will rise an error like 'call to this must be first statement'.

EX:

```
1)  class A
2)  {
3)     A()
4)     {
5)        this(10);
6)        this(22.22f);
7)        System.out.println("A-0-arg-con");
8)     }
9)     A(int i)
10)    {
11)       System.out.println("A-int-param-con");
12)    }
13)    A(float f)
14)    {
15)       System.out.println("A-float-param-con");
16)    }
17) }
18) class Test
19) {
20)    public static void main(String args[])
21)    {
22)       A a=new A();
23)    }
24) }
```

Status: Compilation Error

4) To Return Current Class Object:

To return current class object by using 'this' keyword thn we have to use the followng syntax.

return this;

EX:

```
1)  class A {
2)        A getRef1() {
3)                          A a = new A();
4)                          return a;
5)        }
6)        A getRef2() {
7)                         return this;
8)        }
9)  }
10) class Test {
11)       public static void main(String args[]) {
12)                A a = new A();//[a=abc123]
13)                System.out.println(a);
14)                System.out.println();
15)                System.out.println(a.getRef1());//[abc123.getRef1()]
16)                System.out.println(a.getRef1());//[abc123.getRef1()]
17)                System.out.println(a.getRef1());//[abc123.getRef1()]
18)                System.out.println();
19)                System.out.println(a.getRef2());//[abc123.getRef2()]
20)                System.out.println(a.getRef2());//[abc123.getRef2()]
21)                System.out.println(a.getRef2());//[abc123.getRef2()]
22)     }
23) }
```

OUTPUT:

```
A@5e3a78ad

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

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

In the above program, 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 new object reference value every time. This approach will increase no. of objects in Java application, it will not provide Objects reusability, it will provide object duplication, it is not suggestible in application development.

In the above program, 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 have called 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 byte code 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.

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". If we access static variable by using a reference variable contains null value then JVM will not rise any exception.
In Java applications, static variables must be declared as class level variables only, they never be declared as local variables.
In Java applications, static variable values will be stored in method area, not in Stack memory and not in Heap Memory.
In Java applications, to access current class static variables we can use "this" keyword.

EX:

```
1)  class A {
2)      static int i=10;
3)      int j = 10;
4)      void m1() {
5)              //static int k=30;--->error
6)              System.out.println("m1-A");
7)              System.out.println(this.i);
8)      }
9)  }
10) class Test {
11)      public static void main(String args[]) {
12)              A a = new A();
13)              System.out.println(a.i);
14)              System.out.println(A.i);
15)              a.m1();
16)              A a1 = null;
17)              //System.out.println(a1.j);--->NullPointerException
18)              System.out.println(a1.i);
19)     }
20) }
```

OUTPUT:

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

EX:

```
1)  class A {
2)     static int i=10;
3)     int j=10;
4)  }
5)  class Test {
6)     public static void main(String args[]) {
7)        A a1 = new A();
8)        System.out.println(a1.i+"   "+a1.j);
9)        a1.i=a1.i+1;
10)       a1.j=a1.j+1;
11)       System.out.println(a1.i+"   "+a1.j);
12)       A a2 = new A();
13)       System.out.println(a2.i+"   "+a2.j);
14)       a2.i=a2.i+1;
```

```
15)     a2.j=a2.j+1;
16)     System.out.println(a1.i+"  "+a1.j);
17)     System.out.println(a2.i+"  "+a2.j);
18)     A a3 = new A();
19)     System.out.println(a3.i+"  "+a3.j);
20)     a3.i=a3.i+1;
21)     a3.j = a3.j+1;
22)     System.out.println(a1.i+"  "+a1.j);
23)     System.out.println(a2.i+"  "+a2.j);
24)     System.out.println(a3.i+"  "+a3.j);
25)   }
26) }
```

static variable recognized and initialized excatly at the time of loading the respective class bytecode to the memory
instance variable are recognized and initialized just before executing the respective class constructor

OUTPUT:

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

static variable is an element from static context
instance variable is an element from instance context

in static variable we can access by creating object for the respective class and using reference varibable by using respective classname
instance variable access only creating object only

nullpointerException is not applicable for static variables
NullPointerException is applicable for instance variables

single copy of static variable shered to all objects
separate copy of instance variable to all objects

static variables are able to provide more shareability
instance variables are able to provide less sharebility

it is possible to access static variables in both static methods and instance methods
it is possible to access instance variables in only instance methods not possible in static methods

static variable stored in method area
instance variable stored in heap memory

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.

Note: In Java, for a particular, class Byte-Code will be loaded only one time but we can create any number 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.**

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

Static methods will allow only static members of the current class, static methods will not allow non-static members of the current class directly.

Note: 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 are not allowing 'this' keyword in its body but to access current class static methods we are able to use 'this' keyword.

EX:

```
1)  class A {
2)      int I = 10;
3)      static int j = 20;
4)      static void m1() {
5)          System.out.println("m1-A");
6)          System.out.println(j);
7)          //System.out.println(i);---->error
8)          //System.out.println(this.j);---------->error
9)          A a = new A();
10)         System.out.println(a.i);
11)     }
12)     void m2() {
13)         System.out.println("m2-A");
14)         this.m1();
15)     }
16) }
17) class Test {
18)     public static void main(String[] args) {
19)             A a = new A();
20)             a.m1();
21)             a = null;
22)             a.m1();
23)             A.m1();
24)     }
25) }
```

static methods are executed at both load time and runtime in both static context and instance context
instance methods are executed at instance context only not static context

OUTPUT:

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

static methods are accessing using creating Object and classname also
instance methods are accessing ony for creating object class

static methods are allowed only static members of the current class
instance methods are allowed both static and instance members of the class

this keyword is not possible inside static methods
this keyword is possible inside instance methods

NullPointerException is not applicable for static methods , by using refernence variable contains null value
NullPointerException is applicable for instance methods , by using refernence variable contains null valuess

20
10

Q) Is it possible to print a line of text on command prompt without using main() method?

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

EX:

```
1)  class Test {
2)        static int i = m1();
3)        static int m1() {
4)              System.out.println("Welcome to durga software soultions");
5)              System.exit(0);//to terminate the application
6)              return 10;
7)        }
8)  }
```

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 are valid up to JAVA6 version, it is invalid from JAVA7 version, because, In JAVA6 version JVM will load main class bytecode to the memory irrespective of main() method availability. In JAVA7, first, JVM will check whether main() method is existed or not in main class, if main() method is available then only JVM will load main class bytecode to the memory, if main() method is not available in main class then JVM will not load main class bytecode to the memory and JVM will provide 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 are able to allow static members of the current class directly, Static blocks are not allowing non-static members of the current class directly.

Note: If we want to access non-static members of the current class in static block then we must create object for the respective class and we have to use the generated reference variable.

Static blocks are not allowing 'this' keyword in its body.

EX:

```
1)  class A {
2)     int i=10;
3)     static int j=20;
4)     static {
5)        System.out.println("SB-A");
6)        System.out.println(i);//-------->Error
7)        A a = new A();
8)        System.out.println(a.i);
9)        System.out.println(j);
10)       System.out.println(this.j);-------->Error
11)    }
12) }
13) class Test {
14)     public static void main(String[] args) {
15)         A a = new A();
16)     }
17) }
```

OUTPUT:

```
SB-A
10
20
```

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

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.

EX:

```
1)  class Test {
2)      static {
3)          System.out.println("Welcome to DurgaSoftware Soultions");
4)          System.exit(0);//To terminate the programme
5)      }
6)  }
```

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 main class 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 program execution.

NOTE: The above question and answer are valid up to JAVA6 version, they are invalid from JAVA7 version onwards, because, in JAVA6 version JVM will load main class bytecode to the memory without checking main() method availability but in JAVA 7, first, JVM will search for main() method, if it is available then only JVM will load main class bytecode to the memory, if it is not existed then JVM will provide the following Error message.

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

Q)Is it possible to display a line of text on command prompt without using main() method, static variable, static method, static block?

Yes, it is possible to display a line of text on command prompt without using main() method, a static variable, a static method and static block but by using "static Anonymous Inner class of Object class".

EX:

```
1)  class Test {
2)      static Object obj = new Object() {
3)          {
4)              System.out.println("Welcome To durga Software Solutions");
5)              System.exit(0);//TO termiante the application.
6)          }
7)      };
```

```
8)  }
```

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 without using package name every time as fully qualified name.**

- **If we want to access classes and interfaces of a particular package without importing then we must use fully qualified name every time that is we have to use package name along with class names.**

- **A java program without import statement:**
  java.io.BufferedReader br=new java.io.BufferedReader(new java.io.InputStreamReader(System.in));

- **A java program 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 without 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".**

Syntaxes:

1) **import static package_Name.Class_Name_Or_Interface_Name.*;**
   It will import all the static members from the specified class or interface.

2) **import static package_Name.Class_Name_Or_Interface_Name.member_Name;**
   It will import only the specified member from the specified class or interface.

EX:

```
1)  import static java.lang.Thread.*;
2)  import static java.lang.System.out;
3)  class Test {
4)        public static void main(String args[]) {
5)                out.println(MIN_PRIORITY);
6)                out.println(MAX_PRIORITY);
7)                out.println(NORM_PRIORITY);
8)        }
9)  }
```

OUTPUT:

```
1
10
5
```

Static Context / Static Flow of Execution:

In Java, Static Context will be represented in the form of the following 3 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.

EX:

```
1)  class A {
2)      static  {
3)         System.out.println("SB-A");
4)      }
5)      static int i = m1();
6)      static int m1() {
7)          System.out.println("m1-A");
8)          return 10;
9)      }
```

```
10) }
11) class Test {
12)       public static void main(String args[]) {
13)             A a = new A();
14)       }
15) }
```

OUTPUT:

SB-A
m1-A

EX:

```
1)  class A {
2)       static int i = m2();
3)       static int m1() {
4)           System.out.println("m1-A");
5)           return 10;
6)       }
7)       static {
8)          System.out.println("SB-A");
9)       }
10)      static int m2() {
11)          System.out.println("m2-A");
12)          return 20;
13)      }
14)      static int j=m1();
15) }
16) class Test {
17)      public static void main(String args[]) {
18)            A a1 = new A();
19)            A a2 = new A();
20)      }
21) }
```

OUTPUT:

m2-A
SB-A
m1-A

EX:

```
1)  class A {
2)      static int i = m2();
3)      A() {
4)          System.out.println("A-con");
5)      }
6)      int m1() {
7)          System.out.println("m1-A");
8)          return 10;
9)      }
10)     static {
11)         System.out.println("SB-A");
12)     }
13)     int j = m1(); {
14)        System.out.println("IB-A");
15)     }
16)     static int m2() {
17)         System.out.println("m2-A");
18)         return 10;
19)     }
20) }
21) class Test {
22)     public static void main(String args[]) {
23)         A a1 = new A();
24)         A a2 = new A();
25)     }
26) }
```

OUTPUT:

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

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 in order to solve design problems.

EX:

```
1)  class A {
2)      private A() {
3)          System.out.println("A-con");
4)      }
5)      void m1() {
6)          System.out.println("m1-A");
7)      }
8)      static A getRef()//Factory Method
9)      {
10)         A a = new A();
11)         return a;
12)     }
13) }
14) class Test {
15)      public static void main(String args[]) {
16)             A a = A.getRef();
17)             a.m1();
18)      }
19) }
```

OUTPUT:
A-con
m1-A

There are 2 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:

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

EX1:

```
1)  class A {
2)      static A a = null;
3)      private A() {
4)      }
5)      static A getRef() {
6)          if(a == null) {
7)              a = new A();
8)              return a;
9)          }
10)         else {
11)           return a;
12)         }
13)     }
14) }
```

```
15) class Test {
16)       public static void main(String args[]) {
17)             System.out.println(A.getRef());
18)             System.out.println(A.getRef());
19)             System.out.println(A.getRef());
20)       }
21) }
```

OUTPUT: A@a6eb38a
A@a6eb38a
A@a6eb38a

Another Alternative for Singleton Class:

```
1)  class A {
2)       static A a = null;
3)       static {
4)          a = new A();
5)       }
6)       private A() {
7)       }
8)       static A getRef() {
9)            return a;
10)      }
11) }
12) class Test {
13)       public static void main(String args[]) {
14)                 System.out.println(A.getRef());
15)                 System.out.println(A.getRef());
16)                 System.out.println(A.getRef());
17)       }
18) }
```

OUTPUT:
A@a6eb38a
A@a6eb38a
A@a6eb38a

Alternative Logic for Singleton Class:

```
1)  class A {
2)       static A a = new A();
3)       private A() {
4)       }
5)       static A getRef() {
6)            return a;
7)       }
```

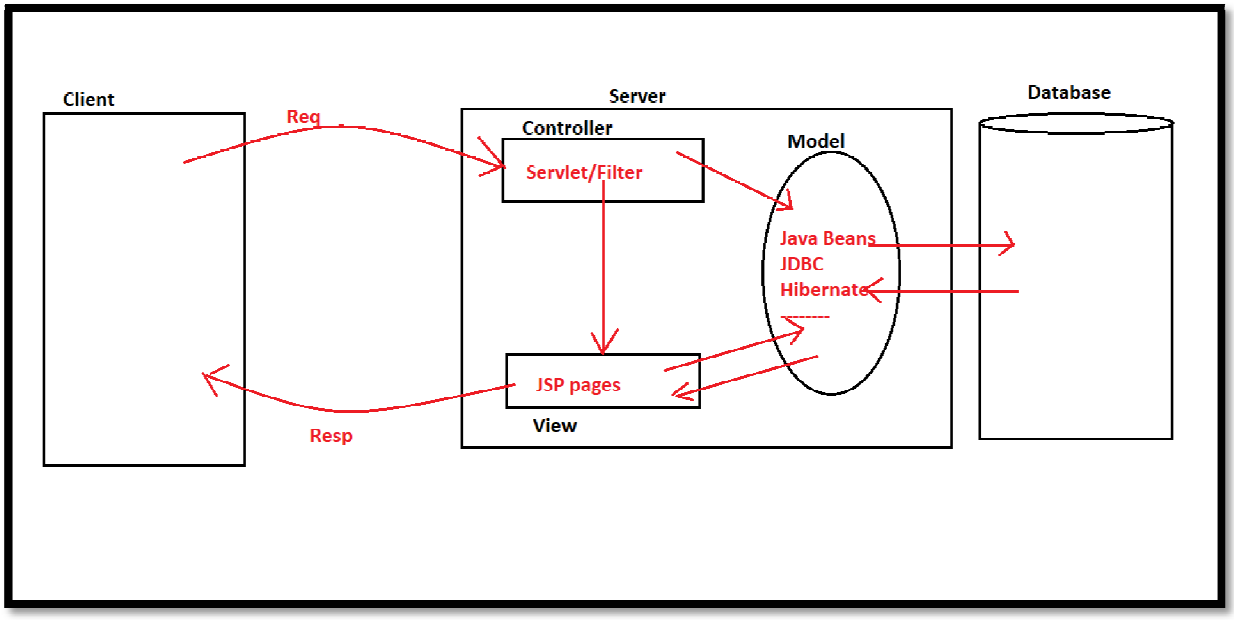
```
8)  }
9)  class Test {
10)         public static void main(String args[]) {
11)                 System.out.println(A.getRef());
12)                 System.out.println(A.getRef());
13)                 System.out.println(A.getRef());
14)         }
15) }
```

servlet lifecycle -
servlet loading - Class c = Class.forName("MyServlet");
servlet Instantiation - MyServlet ms = (MyServlet) c.newInstance();
servlet Initialization -
request processing
Servelt Deinstantiation -

OUTPUT:

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

MVC is a design pattern, it will provide a standard structure to prepare web applications/GUI applications, where in MVC based applications we must use a servlet/ filter [A Java class] as controller, a set of JSP pages/Html pages as view part and a java bean or EJB component or JDBC program or Hibernate program as Model component. As per MVC Arch rules and regulations, only one controller must be provided for application, that is, the class which we used as controller must provide one object, therefore, controller class must be singleton class.



EX1: Struts is MVC based Framework, where "ActionServlet" is controller, so it is Singleton class.

EX2: JSF[Java Server Faces] is MVC based framework, where "FacesServlet" is controller, so it is Singleton class.

Class.forName(--)

Consider the following program

```
1)  class A
2)  {
3)     static
4)     {
5)        System.out.println("Class Loading");
6)     }
7)     A()
8)     {
9)        System.out.println("Object Creating");
10)    }
11) }
12) class Test
13) {
14)    public static void main(String[] args)throws Exception
15)    {
16)       A a=new A();
17)    }
18) }
```

When we access a constructor along with "new" keyword then JVM will perform the following actions automatically.

1.JVM will load the respective class bytecode to the memory.
2.JVM will create object for the loaded class.

As per the requirement, if we want to load class byte code to the memory with out creating object then 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 class name from forName(-) method.
2) JVM will search for its .class file at current location, at java predefined library and at the locations refered by "classpath" environment variable.
3) If the required .class file is not available at all the above locations then JVM will rise an exception like "java.lang.ClassNotFoundException".
4) If the required .class file is available at either of the above locations then JVM will load its bytecode to the memory.
5) After loading class bytecode to the memory, JVM will take metadata of the loaded class like class name, super class details, implemented interfaces details, variables

details, methods details,.... and JVM will store them by creating java.lang.Class object in heap memory and JVM will return the generated Class object reference value from Class.forName(-);

EX:

```
1)  class A
2)  {
3)     static
4)     {
5)        System.out.println("Class Loading");
6)     }
7)     A()
8)     {
9)        System.out.println("Object Creating");
10)    }
11) }
12) class Test
13) {
14)    public static void main(String[] args)throws Exception
15)    {
16)       Class c=Class.forName("A");
17)    }
18) }
```

OUTPUT: Class Loading

After loading class bytecode by using Class.forName(-) method, if we want to create object explicitly then we have to use the following method.

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 goto the loaded class bytecode and JVM will check whether 0-arg and non-private constructor is existed or not.
2) If 0-arg and non-private constructor is existed then JVM will execute that constructor and JVM will create object for the loaded class.
3) If the constructor parameterized constructor without 0-arg constructor then JVM will rise an excewption like "java.lang.InstantiationException".
4) If the constructor is private constrictor then JVM will rise an exception like "java.lang.IllegalAccessException".

Note: If the constructor is both parameterized and 0-arg then JVM will rise "java.lang.InstaitiationException" only.

EX:

```
1)  class A
2)  {
3)     static
4)     {
5)        System.out.println("Class Loading");
6)     }
7)     A()
8)     {
9)        System.out.println("Object Creating");
10)    }
11) }
12) class Test
13) {
14)    public static void main(String[] args)throws Exception
15)    {
16)       Class c=Class.forName("A");
17)       Object obj=c.newInstance();
18)    }
19) }
```

OUTPUT:

Class Loading
Object Creating

In JDBC Appl, we are going to use Driver to map JAVA representations to Database representations and Database representations to Java representations. In Jdbc applications, we must load driver class, not to create object for Driver class, for this, we have to use "Class.forName(-)"

In general, all server side components like Servlets, JSPs, EJBs,... are executed by the server[Container] by following their lifecycle actions like loading, instantiation, innitialization,..... In this context, to perform server side components loading Container will use "Class.forName(-)" method and to perform instantiation lifecycle action container will use "newInstance()" method internally.

Final Keyword:

final is a Java Keyword it can be used to declare constant expressions.
In java applications, 'final' keyword is able to improve security.

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;----> Compilation Error

EX:
for(final int i=0;i<10;i++) ----> Compilation Error
{
System.out.println(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' variable.

2) final Method:

- final method is a Java method, it will not allow method overriding.
- In method overriding, we have to provide same method with different implementation at both super class and at sub class, where super class method never be declared as final irrespective of sub class method final.

EX1:

```
1)  class A {
2)        final void m1() {
3)              System.out.println("m1-A");
4)        }
5)  }
6)  class B extends A {
7)         void m1() {
8)              System.out.println("m1-B");
9)         }
```

```
10) }
11) class Test {
12)         public static void main(String[] args) {
13)               A  a =  new B();
14)               a.m1();
15)         }
16) }
```

Status: Compilation Error.

EX2:

```
1)  class A {
2)       final void m1() {
3)            System.out.println("m1-A");
4)       }
5)  }
6)  class B extends A {
7)          final void m1() {
8)               System.out.println("m1-B");
9)          }
10) }
11) class Test {
12)        public static void main(String[] args) {
13)             A  a =  new B();
14)             a.m1();
15)        }
16) }
```

Status: Compilation Error

EX3:

```
1)  class A {
2)       void m1() {
3)            System.out.println("m1-A");
4)       }
5)  }
6)  class B extends A {
7)         final void m1() {
8)              System.out.println("m1-B");
9)         }
10) }
11) class Test {
12)       public static void main(String[] args) {
13)              A  a =  new B();
14)              a.m1();
```

```
15)     }
16) }
```

Status: No Compilation Error.

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 sub classes may be final.**

EX:

```
1)  final class A
2)  {
3)  }
4)  class B extends A
5)  {
6)  }
```

Status:Invalid

EX:

```
1)  final class A
2)  {
3)  }
4)  final class B extends A
5)  {
6)  }
```

Status:Invalid

EX:

```
1)  class A
2)  {
3)  }
4)  final class B extends A
5)  {
6)  }
```

Status:Valid

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

EX:

In System Class:

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

In Thread Class:

```
public static final int MIN_PRIORITY=1;
public static final int NORM_PRIORITY=5;
public static final int MAX_PRIORITY=10;
```

EX:

```
1)  class User_Status {
2)          public static final String AVAILABLE="Available";
3)          public static final String BUSY="Busy";
4)          public static final String IDLE="Idle";
5)  }
6)  class Test {
7)          public static void main(String args[]) {
8)              System.out.println(User_Status.AVAILABLE);
9)              System.out.println(User_Status.BUSY);
10)             System.out.println(User_Status.IDLE);
11)         }
12) }
```

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 in 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 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.

Syntax:

```
[Access_modifier] enum Enum_Name
{
  ----- List of constants-----
}
```

EX:

```
1)  enum User_Status {
2)        AVAILABLE,BUSY,IDLE;
3)  }
4)  class Test {
5)        public static void main(String args[]) {
6)              System.out.println(User_Status.AVAILABLE);
7)              System.out.println(User_Status.BUSY);
8)              System.out.println(User_Status.IDLE);
9)        }
10) }
```

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

```
final class User_Status 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".

NOTE: In Java applications,it is possible to implement inheritance between two classes but it is not possible to implement inheritance between two "enums", because, bydefault, enums are final classses.

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

EX

```
1)  enum Apple {
2)        A(500),B(250),C(100);
3)        int price;
4)        Apple(int price) {
5)            this.price = price;
6)        }
7)        public int getPrice() {
8)            return price;
9)        }
10) }
11) class Test {
12)       public static void main(String args[]) {
13)             System.out.println("A-Grade Apple   :"+Apple.A.getPrice());
14)             System.out.println("B-Grade Apple   :"+Apple.B.getPrice());
15)             System.out.println("C-Grade Apple   :"+Apple.C.getPrice());
16)       }
17) }
```

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

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

```
1)  final class Apple extends Enum {
2)        public static final Apple A = new Apple(500);
3)        public static final Apple B = new Apple(250);
4)        public static final Apple C = new Apple(100);
5)        int price;
6)        Apple(int price) {
7)            this.price = price;
```

```
8)          }
9)          public int getPrice() {
10)                 return price;
11)         }
12) ----
13) }
```

EX:

```
1)  enum Book {
2)          A(500,250),B(300,150),C(200,100);
3)          int no_of_pages;
4)          int cost;
5)          Book(int no_of_pages, int cost) {
6)               this.no_of_pages = no_of_pages;
7)               this.cost = cost;
8)           }
9)           public void getBookDetails() {
10)                System.out.println(no_of_pages+"--------->"+cost);
11)          }
12) }
13) class Test {
14)       public static void main(String args[]) {
15)               System.out.println("Durga Books Store");
16)               System.out.println("------------------");
17)               System.out.println("No of Pages   Cost");
18)               System.out.println("------------------");
19)               Book.A.getBookDetails();
20)               Book.B.getBookDetails();
21)               Book.C.getBookDetails();
22)       }
23) }
```

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):

```
1)  final class Book extends Enum {
2)          public static final Book A=new Book(500,250);
3)          public static final Book B=new Book(300,150);
4)          public static final Book C=new Book(200,100);
5)          int cost;
6)          int no_of_pages;
7)          Book(int no_of_pages,int cost) {
8)                  this.no_of_pages = no_of_pages;
9)                  this.cost = cost;
10)         }
11)         public void getBookDetails() {
12)                 System.out.println(no_of_pages+"--------->"+cost);
13)         }
14)          ----
15) }
```

Importance of main() method in Java:

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

1) To define application logic in Java program.
2) To define starting point and ending point for the applications execution.

Syntax:

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

Note: Main() method is not predefined method and it is not 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 in order 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 up to the main class only ,not to JVM.

Case-2:

If main() method is declared as "<default>" then main() method will be available up to 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 up to the package where main class is existed and up to child classes available in other packages but not to the JVM.

Case-4:

To make available main() method to JVM, only one possibility we have to take that is "public", where public members are available throughout 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"?

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

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 without 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?

In Java applications, there are 3 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:

```
1)  class Test
2)  {
3)    int i=10;
4)    int j=20;
5)    void add()
6)   {
7)     System.out.println(i+j);
8)   }
9)  }
```

2) Dynamic Input:

Providing Input to the Java program at runtime.

```
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.

1) JVM will read all the command line input at the time of reading main class name from command prompt.
2) JVM will store all the command line inputs in the form of String[]
3) 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 method is required parameters in order 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?

In general, from application to application or from developer to developer the types of command line input may be varied, here even developers are providing different types of command line input, still, 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?

In general, from application to application and from developer to developer number of command line inputs may be varied, here even developers are providing variable number 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 allow multiple number of command line input.

EX:

```
1)  class Test {
2)      public static void main(String args[]) {
3)          for(int i=0;i<args.length;i++) {
4)              System.out.println(args[i]);
5)          }
6)  }
```

OUTPUT:

```
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
```

EX:

```
1)  class Test
2)  {
3)     public static void main(String[] args) throws Exception
4)     {
5)        String val1=args[0];
6)        String val2=args[1];
7)        int fval=Integer.parseInt(val1);
8)        int sval=Integer.parseInt(val2);
9)        System.out.println("ADD  :"+(fval+sval));
10)       System.out.println("SUB  :"+(fval-sval));
11)       System.out.println("MUL  :"+(fval*sval));
12)    }
13) }
```

OUTPUT:

```
D:\javaapps>javac Test.java
D:\javaapps>java Test 10 5
 ADD  :15
 SUB  :5
 MUL  :50
```

NOTE: 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)----> Invalid
4) public static void main(String[][] args)-->Invalid
5) public static void main(String args[])---> Valid
6) public static void main(String []args)---> Valid
7) public static void main(string[] args)---> Invalid
8) public static void Main(String[] args)---> Invalid
9) public static int main(String[] args)----> Invalid
10) public final void main(String[] args)---> Invalid
11) public void main(String[] args)---------> Invalid
12) static void main(String[] args)---------> Invalid
13) static public void main(String[] args)--> Valid
14) public static void main(String ... args)-> Valid

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

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

EX: File Name : D:\java7\abc.java

```
1)  class A {
2)        public static void main(String args[]) {
3)            System.out.println("main()-A");
4)        }
5)  }
6)  class B {
7)        public static void main(String args[]) {
8)            System.out.println("main()-B");
9)        }
10) }
```

OUTPUT:

```
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: FIle name : D:\java7\abc.java

```
1)  class A {
2)          public static void main(String args[]) {
3)                  System.out.println("main()-A");
4)                  String[] str = {"AAA","BBB","CCC"};
5)                  B.main(str);
6)                  B.main(args);
7)          }
8)  }
9)  class B {
10)         public static void main(String args[]) {
11)             System.out.println("main()-B");
12)         }
13) }
```

OUTPUT:

```
D:\java7>javac abc.java
D:\java7>java A
 main()-A
 main()-B
 main()-B
```

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

Yes, 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 not possible.

EX:

```
1)  class Test
2)  {
3)     public static void main(String[] args)
4)     {
5)        System.out.println("String[]-param-main()");
6)     }
```

```
7)    public static void main(int[] args)
8)    {
9)       System.out.println("int[]-param-main()");
10)   }
11)   public static void main(float[] args)
12)   {
13)      System.out.println("float[]-param-main()");
14)   }
15) }
```

OUTPUT: String[]-param-main

Q . it is possible to main() method inside inner class ?
yes it is possible provide main() method inside inner class but the respective inner class must be static inner class

Q. main() method preefined method or user defined method ?
main() method is not user defined or not predefied method but it is a convensional methpod with fixed prototype and with user defined implementation

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 3 types of relationships between entities.

1) **Has-A Relationship**
2) **IS-A Relationship**
3) **USE-A Relationship**

Q)What is the difference between *HAS-A Relationship* and *IS-A Relationship?*

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

IS-A Relationship is able to define inheritance between entity classes, it will improve "Code Reusability" in java applications.

Associations in JAVA:

There are four types of associations between entities

1) **One-To-One Association**
2) **One-To-Many Association**
3) **Many-To-One Association**
4) **Many-To-Many Association**

To achieve associations between entities, we have to declare either single reference or array of reference variables 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 relation between entities, where one instance of an entity should be mapped with exactly one instance of another entity.

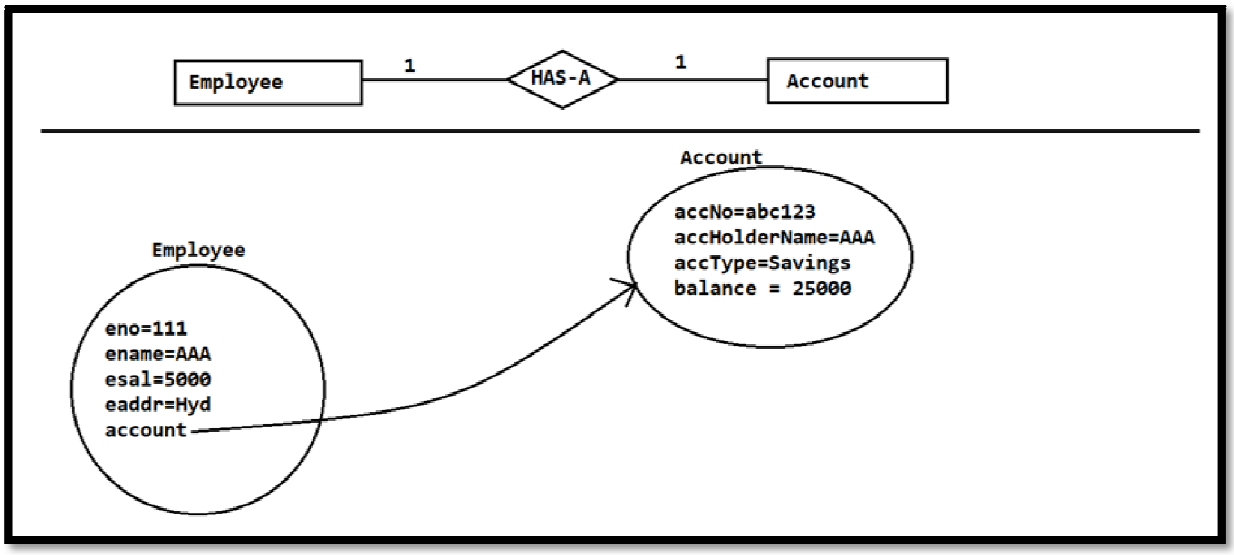
EX: Every employee should have exactly one Account.

```
1)  class Account {
2)          String accNo;
3)          String accName;
4)          String accType;
5)          Account(String accNo,String accName,String accType) {
6)                      this.accNo = accNo;
7)                      this.accName = accName;
8)                      this.accType = accType;
9)          }
10) }
11) class Employee {
12)          String eid;
13)          String ename;
14)          String eaddr;
15)          Account acc;
16)          Employee(String eid, String ename, String eaddr, Account acc) {
17)                  this.eid = eid;
18)                  this.ename = ename;
19)                  this.eaddr = eaddr;
20)                  this.acc = acc;
21)          }
22)          public void getEmployee() {
23)                  System.out.println("Employee Details");
24)                  System.out.println("----------------");
25)                  System.out.println("Employee Id     :"+eid);
26)                  System.out.println("Employee Name   :"+ename);
27)                  System.out.println("Employee Address :"+eaddr);
```

```
28)                 System.out.println();
29)                 System.out.println("Account Details");
30)                 System.out.println("----------------");
31)                 System.out.println("Account Number    :"+acc.accNo);
32)                 System.out.println("Account Name      :"+acc.accName);
33)                 System.out.println("Account Type      :"+acc.accType);
34)           }
35) }
36) class OneToOneEx {
37)           public static void main(String args[]) {
38)                 Account acc = new Account("abc123","Durga N","Savings");
39)                 Employee emp = new Employee("E-111","Durga","Hyd",acc);
40)                 emp.getEmployee();
41)           }
42) }
```

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
```



Dependency Injection:

--> The process of injecting contained object in Container object is called as Dependency Injection.

There are three types of Dependency Injection.
1. Constructor Dependency injection.
2. Setter Method Dependency Injection.
3. Interfaces Dependency Injection.

1. Constructor Dependency injection:

The process of injecting dependent object to the container object through a constructor then that type of Dependency Injection is called as "Constructor Dependency Injection".

Account.java

```
1)  package com.durgasoft.entities;
2)
3)  public class Account {
4)     String accNo;
5)     String accHolderName;
6)     String accType;
7)     int accBalance;
8)
9)     public Account(String accNo, String accHolderName, String accType, int accBalan
    ce) {
10)       this.accNo = accNo;
11)       this.accHolderName = accHolderName;
12)       this.accType = accType;
13)       this.accBalance = accBalance;
14)    }
15) }
```

Employee.java

```
1)  package com.durgasoft.entities;
2)
3)  public class Employee {
4)     int eno;
5)     String ename;
6)     float esal;
7)     String eaddr;
8)
9)     Account acc;
```

```
10)  public Employee(int eno, String ename, float esal, String eaddr, Account acc) {
11)      this.eno = eno;
12)      this.ename = ename;
13)      this.esal = esal;
14)      this.eaddr = eaddr;
15)      this.acc = acc;
16)   }
17)
18)   public void getEmpDetails() {
19)      System.out.println("Employee Details");
20)      System.out.println("-----------------------");
21)      System.out.println("Employee Number    : "+eno);
22)      System.out.println("Employee Name      : "+ename);
23)      System.out.println("Employee Salary    : "+esal);
24)      System.out.println("Employee Address   : "+eaddr);
25)      System.out.println();
26)      System.out.println("Account Details");
27)      System.out.println("-------------------------");
28)      System.out.println("Account Number     : "+acc.accNo);
29)      System.out.println("Account Holder Name : "+acc.accHolderName);
30)      System.out.println("Account Type       : "+acc.accType);
31)      System.out.println("Account Balance    : "+acc.accBalance);
32)   }
33) }
```

Test.java

```
1)  package com.durgasoft.core;
2)
3)  import com.durgasoft.entities.Employee;
4)  import com.durgasoft.entities.Account;
5)  public class Test {
6)
7)     public static void main(String[] args) {
8)        Account acc = new Account("abc123", "Durga", "Savings", 25000);
9)        Employee emp = new Employee(111, "Durga", 15000, "Hyd", acc);
10)       emp.getEmpDetails();
11)    }
12) }
```

2. Setter Method Dependency Injection:

The process of injecting the dependenct object into the Container object then it is called as "Setter Method Dependency Injection".

EX: Account.java

```
1)  package com.durgasoft.entities;
2)
3)  public class Account {
4)     private String accNo;
5)     private String accHolderName;
6)     private String accType;
7)     private int accBalance;
8)
9)     public String getAccNo() {
10)       return accNo;
11)    }
12)    public void setAccNo(String accNo) {
13)       this.accNo = accNo;
14)    }
15)    public String getAccHolderName() {
16)       return accHolderName;
17)    }
18)    public void setAccHolderName(String accHolderName) {
19)       this.accHolderName = accHolderName;
20)    }
21)    public String getAccType() {
22)       return accType;
23)    }
24)    public void setAccType(String accType) {
25)       this.accType = accType;
26)    }
27)    public int getAccBalance() {
28)       return accBalance;
29)    }
30)    public void setAccBalance(int accBalance) {
31)       this.accBalance = accBalance;
32)    }
33) }
```

Employee.java

```
1)  package com.durgasoft.entities;
2)
3)  public class Employee {
4)     private int eno;
5)     private String ename;
6)     private float esal;
7)     private String eaddr;
8)
9)     private Account acc;
10)
11)    public int getEno() {
12)       return eno;
13)    }
14)    public void setEno(int eno) {
15)       this.eno = eno;
16)    }
17)    public String getEname() {
18)       return ename;
19)    }
20)    public void setEname(String ename) {
21)       this.ename = ename;
22)    }
23)    public float getEsal() {
24)       return esal;
25)    }
26)    public void setEsal(float esal) {
27)       this.esal = esal;
28)    }
29)    public String getEaddr() {
30)       return eaddr;
31)    }
32)    public void setEaddr(String eaddr) {
33)       this.eaddr = eaddr;
34)    }
35)    public Account getAcc() {
36)       return acc;
37)    }
38)    public void setAcc(Account acc) {
39)       this.acc = acc;
40)    }
41)
42)    public void getEmpDetails() {
43)       System.out.println("Employee Details");
```

```
44)       System.out.println("--------------------------");
45)       System.out.println("Employee Number    : "+eno);
46)       System.out.println("Employee Name      : "+ename);
47)       System.out.println("Employee Salary    : "+esal);
48)       System.out.println("Employee Address   : "+eaddr);
49)       System.out.println();
50)
51)       System.out.println("Account Details");
52)       System.out.println("--------------------------");
53)       System.out.println("Account Number       : "+acc.getAccNo());
54)       System.out.println("Account Holder Name  : "+acc.getAccHolderName());
55)       System.out.println("Account Type         : "+acc.getAccType());
56)       System.out.println("Account Balance      : "+acc.getAccBalance());
57)    }
58) }
```

Test.java

```
1)  package com.durgasoft.test;
2)
3)  import com.durgasoft.entities.Account;
4)  import com.durgasoft.entities.Employee;
5)
6)  public class Test {
7)
8)     public static void main(String[] args) {
9)        Account acc = new Account();
10)       acc.setAccNo("abc123");
11)       acc.setAccHolderName("Durga");
12)       acc.setAccType("Savings");
13)       acc.setAccBalance(25000);
14)
15)       Employee emp = new Employee();
16)       emp.setEno(111);
17)       emp.setEname("Durga");
18)       emp.setEsal(10000);
19)       emp.setEaddr("Hyd");
20)       emp.setAcc(acc);
21)
22)       emp.getEmpDetails();
23)
24)    }
25)
26) }
```

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.

```
1)  class Employee {
2)          String eid;
3)          String ename;
4)          String eaddr;
5)          Employee(String eid, String ename, String eaddr) {
6)                  this.eid = eid;
7)                  this.ename = ename;
8)                  this.eaddr = eaddr;
9)          }
10) }
11) class Department {
12)          String did;
13)          String dname;
14)          Employee[] emps;
15)          Department(String did, String dname, Employee[] emps) {
16)                this.did = did;
17)                this.dname = dname;
18)                this.emps = emps;
19)          }
20)          public void getDepartmentDetails() {
21)                  System.out.println("Department Details");
22)                  System.out.println("--------------------");
23)                  System.out.println("Department Id :"+did);
24)                  System.out.println("Department Name:"+dname);
25)                  System.out.println();
26)                  System.out.println("EID ENAME EADDR");
27)                  System.out.println("----------------");
28)                  for(int i=0; i<emps.length; i++) {
29)                        Employee e = emps[i];
30)                        System.out.println(e.eid+"  "+e.ename+"  "+e.eaddr);
31)                  }
32)          }
33) }
34) class OneToManyEx {
35)          public static void main(String args[]) {
36)                  Employee e1=new Employee("E-111","AAA","Hyd");
37)                  Employee e2=new Employee("E-222","BBB","Hyd");
38)                  Employee e3=new Employee("E-333","CCC","Hyd");
39)                  Employee[] emps=new Employee[3];
```

```
40)                     emps[0]=e1;
41)                     emps[1]=e2;
42)                     emps[2]=e3;
43)                     Department dept=new Department("D-111","Admin",emps);
44)                     dept.getDepartmentDetails();
45)             }
46) }
```

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
```

EX:

Employee.java

```
1)  package com.durgasoft.entities;
2)
3)  public class Employee {
4)      private int eno;
5)      private String ename;
6)      private float esal;
7)      private String eaddr;
8)
9)      public int getEno() {
10)         return eno;
11)     }
12)     public void setEno(int eno) {
13)         this.eno = eno;
14)     }
15)     public String getEname() {
16)         return ename;
17)     }
18)     public void setEname(String ename) {
19)         this.ename = ename;
20)     }
```

```
21)   public float getEsal() {
22)      return esal;
23)   }
24)   public void setEsal(float esal) {
25)      this.esal = esal;
26)   }
27)   public String getEaddr() {
28)      return eaddr;
29)   }
30)   public void setEaddr(String eaddr) {
31)      this.eaddr = eaddr;
32)   }
33) }
```

Department.java

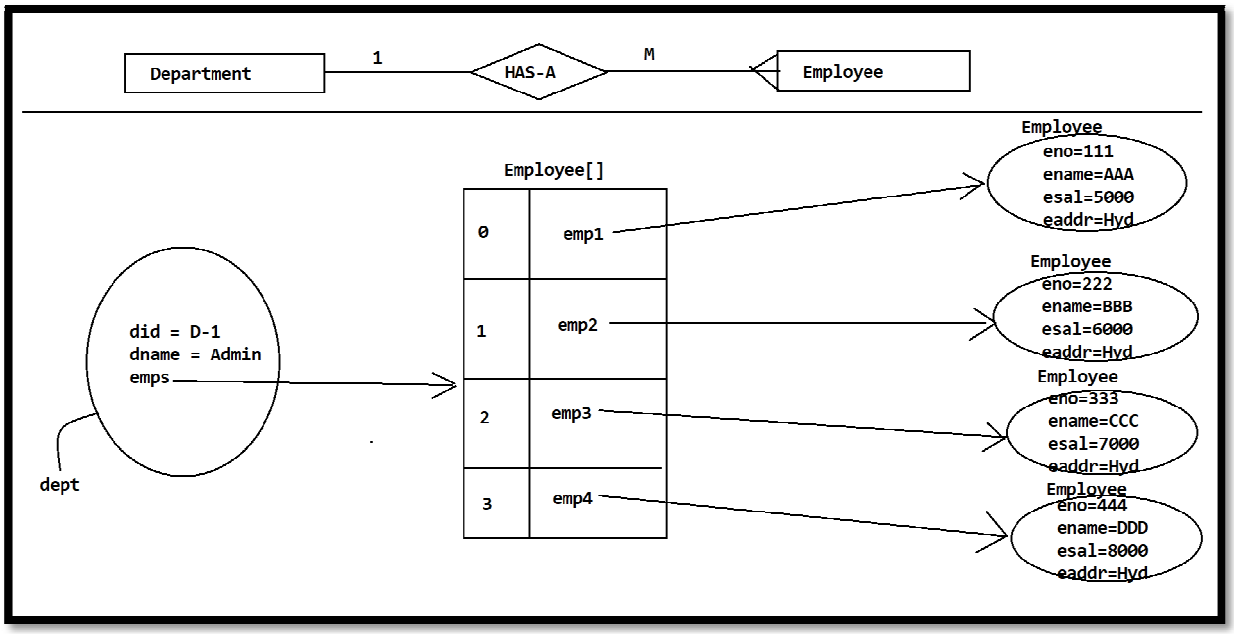
```
1)  package com.durgasoft.entities;
2)
3)  public class Department {
4)     private String did;
5)     private String dname;
6)     private Employee[] emps;
7)
8)     public String getDid() {
9)        return did;
10)    }
11)    public void setDid(String did) {
12)       this.did = did;
13)    }
14)    public String getDname() {
15)       return dname;
16)    }
17)    public void setDname(String dname) {
18)       this.dname = dname;
19)    }
20)    public Employee[] getEmps() {
21)       return emps;
22)    }
23)    public void setEmps(Employee[] emps) {
24)       this.emps = emps;
25)    }
26)
27)    public void getDeptDetails() {
28)       System.out.println("Department Details");
29)       System.out.println("-----------------------");
```

```
30)       System.out.println("Department Id     : "+did);
31)       System.out.println("Department Name   : "+dname);
32)       System.out.println();
33)       System.out.println("ENO\tENAME\tESAL\tEADDR");
34)       System.out.println("----------------------------");
35)       for(Employee emp: emps) {
36)          System.out.print(emp.getEno()+"\t");
37)          System.out.print(emp.getEname()+"\t");
38)          System.out.print(emp.getEsal()+"\t");
39)          System.out.print(emp.getEaddr()+"\n");
40)       }
41)
42)    }
43) }
```

Test.java

```
1)  package com.durgasoft.test;
2)
3)  import com.durgasoft.entities.Department;
4)  import com.durgasoft.entities.Employee;
5)
6)  public class Test {
7)
8)     public static void main(String[] args) {
9)        Employee emp1 = new Employee();
10)       emp1.setEno(111);
11)       emp1.setEname("AAA");
12)       emp1.setEsal(5000);
13)       emp1.setEaddr("Hyd");
14)
15)       Employee emp2 = new Employee();
16)       emp2.setEno(222);
17)       emp2.setEname("BBB");
18)       emp2.setEsal(6000);
19)       emp2.setEaddr("Chennai");
20)
21)       Employee emp3 = new Employee();
22)       emp3.setEno(333);
23)       emp3.setEname("CCC");
24)       emp3.setEsal(7000);
25)       emp3.setEaddr("Delhi");
26)
27)       Employee emp4 = new Employee();
28)       emp4.setEno(444);
```

```
29)       emp4.setEname("DDD");
30)       emp4.setEsal(8000);
31)       emp4.setEaddr("Pune");
32)
33)       Employee[] emps = {emp1, emp2, emp3, emp4};
34)
35)       Department dept = new Department();
36)       dept.setDid("D-111");
37)       dept.setDname("Admin");
38)       dept.setEmps(emps);
39)       dept.getDeptDetails();
40)
41)   }
42) }
```



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 Students have joined with a single branch.

```
1)  class Branch {
2)        String bid;
3)        String bname;
4)        Branch(String bid, String bname) {
5)              this.bid = bid;
6)              this.bname = bname;
7)        }
```

```
8)  }
9)  class Student {
10)         String sid;
11)         String sname;
12)         String saddr;
13)         Branch branch;
14)         Student(String sid, String sname, String saddr, Branch branch) {
15)                 this.sid = sid;
16)                 this.sname = sname;
17)                 this.saddr = saddr;
18)                 this.branch = branch;
19)         }
20)         public void getStudentDetails() {
21)                 System.out.println("Student Details");
22)                 System.out.println("----------------");
23)                 System.out.println("Student Id    :"+sid);
24)                 System.out.println("Student name  :"+sname);
25)                 System.out.println("Student Address:"+saddr);
26)                 System.out.println("Branch Id   :"+branch.bid);
27)                 System.out.println("Branch Name :"+branch.bname);
28)                 System.out.println();
29)     }
30) }
31) class ManyToOneEx {
32)         public static void main(String args[]) {
33)                 Branch branch=new Branch("B-111","CS");
34)                 Student std1=new Student("S-111","AAA","Hyd",branch);
35)                 Student std2=new Student("S-222","BBB","Hyd",branch);
36)                 Student std3=new Student("S-333","CCC","Hyd",branch);
37)                 std1.getStudentDetails();
38)                 std2.getStudentDetails();
39)                 std3.getStudentDetails();
40)         }
41) }
```

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
```

EX:

Branch.java

```
1)  package com.durgasoft.entities;
2)
3)  public class Branch {
4)     private String bid;
5)     private String bname;
6)
7)     public String getBid() {
8)        return bid;
9)     }
10)    public void setBid(String bid) {
11)       this.bid = bid;
12)    }
13)    public String getBname() {
14)       return bname;
15)    }
16)    public void setBname(String bname) {
17)       this.bname = bname;
18)    }
19) }
```

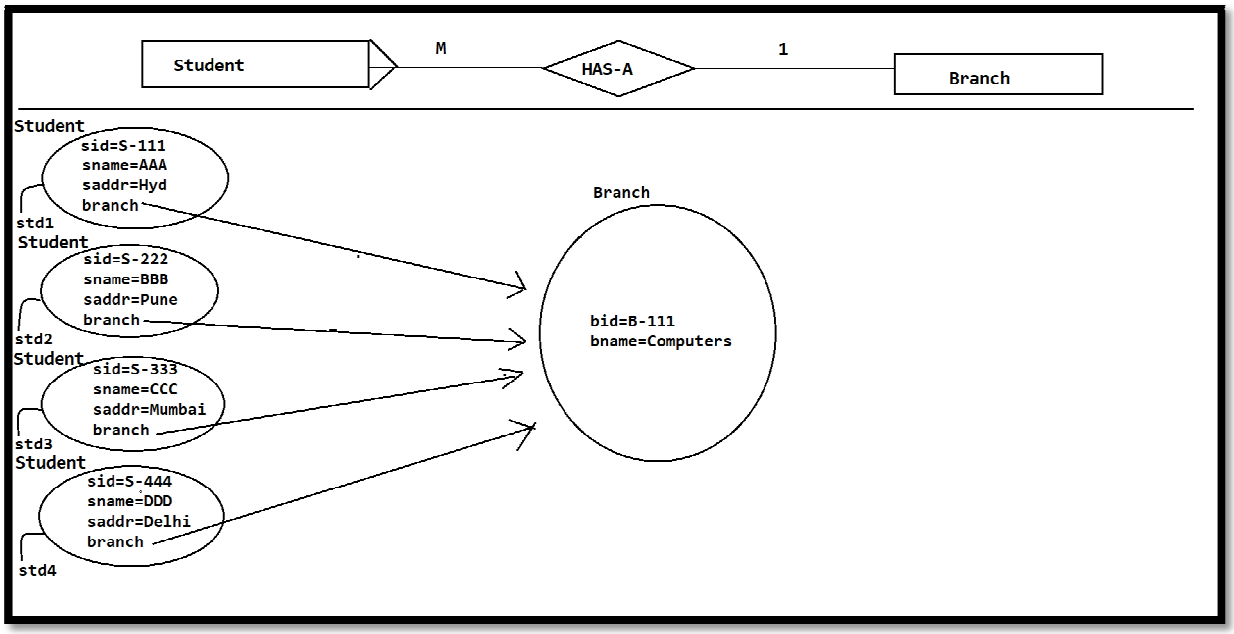
Student.java

```
1)  package com.durgasoft.entities;
2)
3)  public class Student {
4)     private String sid;
5)     private String sname;
6)     private String saddr;
7)     private Branch branch;
8)
9)     public String getSid() {
10)       return sid;
11)    }
12)    public void setSid(String sid) {
13)       this.sid = sid;
14)    }
15)    public String getSname() {
16)       return sname;
17)    }
18)    public void setSname(String sname) {
19)       this.sname = sname;
20)    }
21)    public String getSaddr() {
22)       return saddr;
23)    }
24)    public void setSaddr(String saddr) {
25)       this.saddr = saddr;
26)    }
27)    public Branch getBranch() {
28)       return branch;
29)    }
30)    public void setBranch(Branch branch) {
31)       this.branch = branch;
32)    }
33)
34)    public void getStudentDetails() {
35)       System.out.println("Student Details");
36)       System.out.println("----------------------");
37)       System.out.println("Student Id     : "+sid);
38)       System.out.println("Student Name   : "+sname);
39)       System.out.println("Student Address : "+saddr);
40)       System.out.println("Branch Id      : "+branch.getBid());
41)       System.out.println("Branch Name    : "+branch.getBname());
42)    }
43) }
```

Test.java

```
1)  package com.durgasoft.test;
2)
3)  import com.durgasoft.entities.Branch;
4)  import com.durgasoft.entities.Student;
5)
6)  public class Test {
7)
8)     public static void main(String[] args) {
9)        Branch branch = new Branch();
10)       branch.setBid("B-111");
11)       branch.setBname("Computers");
12)
13)       Student std1 = new Student();
14)       std1.setSid("S-111");
15)       std1.setSname("AAA");
16)       std1.setSaddr("Hyd");
17)       std1.setBranch(branch);
18)       std1.getStudentDetails();
19)       System.out.println();
20)
21)       Student std2 = new Student();
22)       std2.setSid("S-222");
23)       std2.setSname("BBB");
24)       std2.setSaddr("Chennai");
25)       std2.setBranch(branch);
26)       std2.getStudentDetails();
27)       System.out.println();
28)
29)       Student std3 = new Student();
30)       std3.setSid("S-333");
31)       std3.setSname("CCC");
32)       std3.setSaddr("Pune");
33)       std3.setBranch(branch);
34)       std3.getStudentDetails();
35)       System.out.println();
36)
37)       Student std4 = new Student();
38)       std4.setSid("S-444");
39)       std4.setSname("DDD");
40)       std4.setSaddr("Delhi");
41)       std4.setBranch(branch);
42)       std4.getStudentDetails();
43)    }
```

```
44) }
```



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.

EX:

```
1)  class Course {
2)          String cid;
3)          String cname;
4)          int ccost;
5)          Course(String cid,String cname,int ccost) {
6)                  this.cid = cid;
7)                  this.cname = cname;
8)                  this.ccost = ccost;
9)          }
10) }
11) class Student {
12)          String sid;
13)          String sname;
14)          String saddr;
15)          Course[] crs;
16)          Student(String sid,String sname,String saddr,Course[] crs) {
17)                  this.sid=sid;
18)                  this.sname=sname;
19)                  this.saddr=saddr;
```

```
20)                 this.crs=crs;
21)         }
22)         public void getStudentDetails() {
23)                 System.out.println("Student Details");
24)                 System.out.println("----------------");
25)                 System.out.println("Student Id    :"+sid);
26)                 System.out.println("Student name  :"+sname);
27)                 System.out.println("Student Address:"+saddr);
28)                 System.out.println("CID  CNAME CCOST");
29)                 System.out.println("---------------------");
30)                 for(int i=0;i<crs.length;i++) {
31)                     Course c=crs[i];
32)                     System.out.println(c.cid+"  "+c.cname+"  "+c.ccost);
33)                 }
34)                 System.out.println();
35)         }
36) }
37) class ManyToManyEx {
38)          public static void main(String[] args) {
39)                     Course c1=new Course("C-111","C",500);
40)                     Course c2=new Course("C-222","C++",1000);
41)                     Course c3=new Course("C-333","JAVA",5000);
42)                     Course[] crs=new Course[3];
43)                     crs[0]=c1;
44)                     crs[1]=c2;
45)                     crs[2]=c3;
46)                     Student std1=new Student("S-111","AAA","Hyd",crs);
47)                     Student std2=new Student("S-222","BBB","Hyd",crs);
48)                     Student std3=new Student("S-333","CCC","Hyd",crs);
49)                     std1.getStudentDetails();
50)                     std2.getStudentDetails();
51)                     std3.getStudentDetails();
52)          }
53) }
```

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
```

EX:

Course.java:

```
1)  package com.durgasoft.entities;
2)
3)  public class Course {
4)     private String cid;
5)     private String cname;
6)     private int ccost;
7)
8)     public String getCid() {
9)        return cid;
10)    }
11)    public void setCid(String cid) {
12)       this.cid = cid;
```

```
13)   }
14)   public String getCname() {
15)      return cname;
16)   }
17)   public void setCname(String cname) {
18)      this.cname = cname;
19)   }
20)   public int getCcost() {
21)      return ccost;
22)   }
23)   public void setCcost(int ccost) {
24)      this.ccost = ccost;
25)   }
26) }
```

Student.java

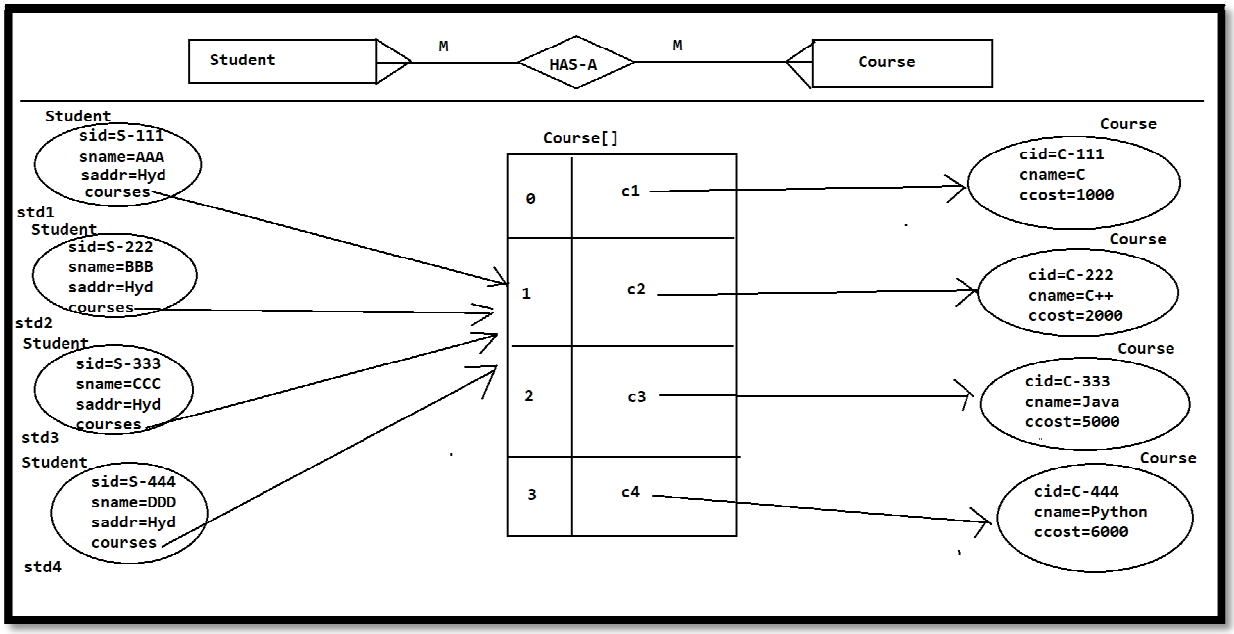
```
1)  package com.durgasoft.entities;
2)
3)  public class Student {
4)     private String sid;
5)     private String sname;
6)     private String saddr;
7)     private Course[] courses;
8)
9)     public String getSid() {
10)       return sid;
11)    }
12)    public void setSid(String sid) {
13)       this.sid = sid;
14)    }
15)    public String getSname() {
16)       return sname;
17)    }
18)    public void setSname(String sname) {
19)       this.sname = sname;
20)    }
21)    public String getSaddr() {
22)       return saddr;
23)    }
24)    public void setSaddr(String saddr) {
25)       this.saddr = saddr;
26)    }
27)    public Course[] getCourses() {
28)       return courses;
```

```
29)   }
30)   public void setCourses(Course[] courses) {
31)      this.courses = courses;
32)   }
33)   public void getStudentDetails() {
34)      System.out.println("Student Details");
35)      System.out.println("----------------------");
36)      System.out.println("Student Id      : "+sid);
37)      System.out.println("Student Name    : "+sname);
38)      System.out.println("Student Address : "+saddr);
39)      System.out.println("CID\tCNAME\tCCOST");
40)      System.out.println("--------------------------");
41)      for(Course c: courses) {
42)         System.out.print(c.getCid()+"\t");
43)         System.out.print(c.getCname()+"\t");
44)         System.out.print(c.getCcost()+"\n");
45)      }
46)      System.out.println();
47)   }
48) }
```

Test.java

```
1)  package com.durgasoft.test;
2)
3)  import com.durgasoft.entities.Course;
4)  import com.durgasoft.entities.Student;
5)
6)  public class Test {
7)
8)     public static void main(String[] args) {
9)        Course c1 = new Course();
10)       c1.setCid("C-111");
11)       c1.setCname("C     ");
12)       c1.setCcost(1000);
13)
14)       Course c2 = new Course();
15)       c2.setCid("C-222");
16)       c2.setCname("C++   ");
17)       c2.setCcost(2000);
18)
19)       Course c3 = new Course();
20)       c3.setCid("C-333");
21)       c3.setCname("Java  ");
22)       c3.setCcost(3000);
```

```
23)
24)         Course c4 = new Course();
25)         c4.setCid("C-444");
26)         c4.setCname("Python");
27)         c4.setCcost(1000);
28)
29)         Course[] courses = {c1, c2, c3, c4};
30)
31)         Student std1 = new Student();
32)         std1.setSid("S-111");
33)         std1.setSname("AAA");
34)         std1.setSaddr("Hyd");
35)         std1.setCourses(courses);
36)
37)         Student std2 = new Student();
38)         std2.setSid("S-222");
39)         std2.setSname("BBB");
40)         std2.setSaddr("Hyd");
41)         std2.setCourses(courses);
42)
43)         Student std3 = new Student();
44)         std3.setSid("S-333");
45)         std3.setSname("CCC");
46)         std3.setSaddr("Hyd");
47)         std3.setCourses(courses);
48)
49)         Student std4 = new Student();
50)         std4.setSid("S-444");
51)         std4.setSname("DDD");
52)         std4.setSaddr("Hyd");
53)         std4.setCourses(courses);
54)
55)         std1.getStudentDetails();
56)         std2.getStudentDetails();
57)         std3.getStudentDetails();
58)         std4.getStudentDetails();
59)     }
60) }
```



In java applications, Associations are existed in the following two forms.

1) **Aggregation**
2) **Composition**

Q) What is the difference between *Aggregation* and *Composition*?

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

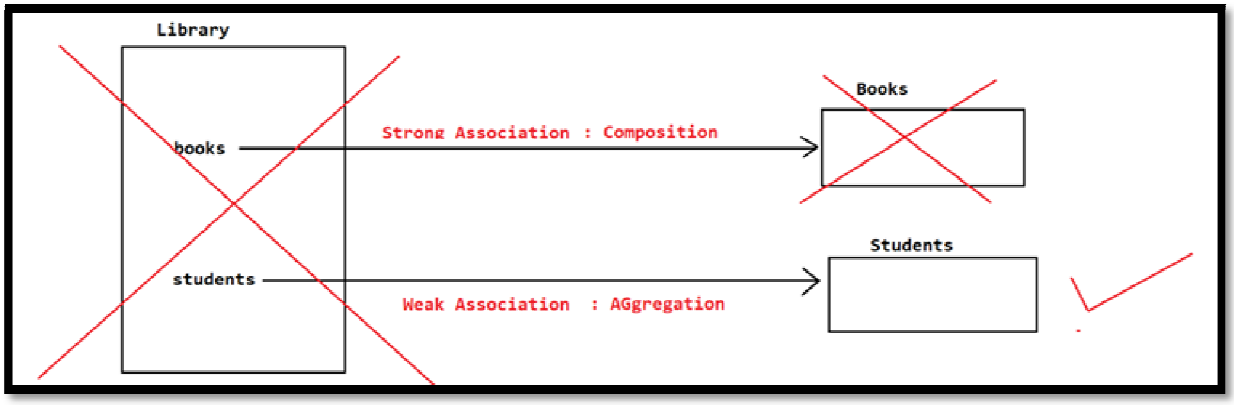
2.In case of Aggregation, if contained object is existed even without container object.
In case of Composition, if contained object is not existed without container object.

3.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.

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



Inheritance in JAVA:

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.

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 number 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 number of sub classes is called as Multiple Inheritance.

Java is not allowing Multiple Inheritance.

On the basis of Single and Multiple inheritances, there are 3 more Inheritances.

1) **Multi-Level Inheritance**
2) **Hierarchical 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. like binery tree

EX:

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

2. Hierarchical 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:

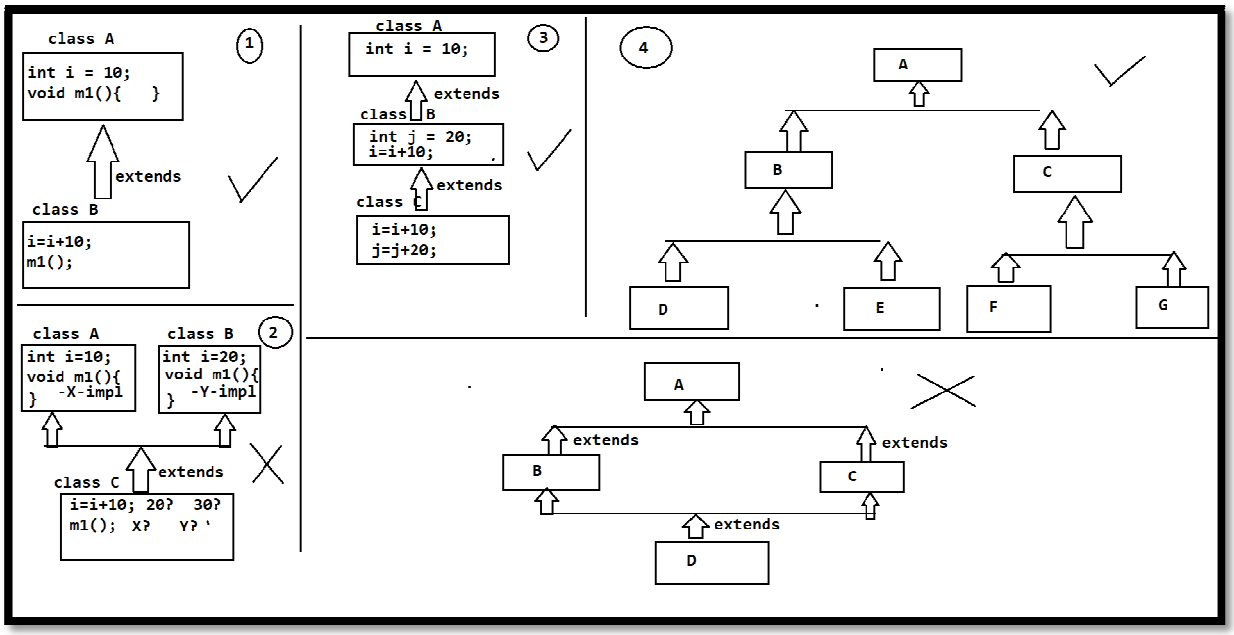
It is the combination of single inheritance and multiple inheritances.

Java is not allowing Hybrid Inheritance.

EX:

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

```
}
class D extends B, C {
}
```



EX:

```
1)  class Employee {
2)          String eid;
3)          String ename;
4)          String eaddr;
5)          public void getEmpDetails() {
6)                  System.out.println("Employee Id :"+eid);
7)                  System.out.println("Employee name :"+ename);
8)                  System.out.println("Employee Address :"+eaddr);
9)          }
10) }
11) class Manager extends Employee {
12)         Manager(String eid1, String ename1, String eaddr1) {
13)                  eid = eid1;
14)                  ename = ename1;
15)                  ename = eaddr1;
16)         }
17)         public void getManagerDetails() {
18)                  System.out.println("manager Details");
19)                  System.out.println("----------------");
20)                  getEmpDetails();
21)     }
22) }
```

```
23) class Accountant extends Employee {
24)         Accountant (String eid1, String ename1, String eaddr1) {
25)                     eid = eid1;
26)                     ename = ename1;
27)                     eaddr = eaddr1;
28)         }
29)         public void getAccountantDetails() {
30)                 System.out.println("Accountant Details");
31)                 System.out.println("--------------------");
32)                 getEmpDetails():
33)         }
34) }
35) class InheritanceEx {
36)         public static void main(String[] args) {
37)                 Manager m = new Manager("E-111","AAA","Hyd");
38)                 m.get ManagerDetails();
39)                 System.out.println();
40)                 Accountant acc = new Accountant ("E-222","BBB","Hyd");
41)                 acc.getAccountantDetails();
42)         }
43) }
```

On Command Prompt:

D:\java7>javac InheritanceEx.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 the above 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 order.

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, that is from super class to sub class ordering.

EX:

```
1)  class A {
2)        static {
3)           System.out.println("SB-A");
4)        }
5)  }
6)  class B extends A {
7)        static {
8)           System.out.println("B-con");
9)        }
10) }
11) class C extends B {
12)       static {
13)          System.out.println("SB-C");
14)       }
15) }
16) class Test {
17)       public static void main(String[] args) {
18)                C c=new C();
19)       }
20) }
```

OUTPUT:

```
SB-A
B-con
SB-C
```

EX:

```
1)  class A {
2)        static {
3)           System.out.println("SB-A");
4)        }
5)        static int m1() {
6)           System.out.println("m1-A");
7)           return 10;
8)        }
9)        static int i=m1();
```

```
10) }
11) class B extends A {
12)         static int j=m2();
13)         static {
14)             System.out.println("SB-B");
15)         }
16)         static int m2() {
17)                 System.out.println("m2-B");
18)                 return 20;
19)         }
20) }
21) class C extends B {
22)         static int m3() {
23)             System.out.println("m3-C");
24)             return 30;
25)         }
26)         static int k=m3();
27)         static {
28)            System.out.println("SB-C");
29)         }
30) }
31) class Test {
32)       public static void main(String[] args) {
33)             C c1=new C();
34)             C c2=new C();
35)       }
36) }
```

OUTPUT:
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. If we provided instance context in both super class and sub class then JVM will execute the provided instance context just before executing the respective class constructor.

EX:

```
1)  class A {
2)      A() {
3)              System.out.println("A-con");
4)      }
5)  }
6)  class B extends A {
7)      B() {
8)              System.out.println("B-con");
9)      }
10) }
11) class C extends B {
12)     C() {
13)             System.out.println("C-con");
14)     }
15) }
16) class Test {
17)     public static void main(String[] args) {
18)             C c = new C();
19)     }
20) }
```

Type text here

OUTPUT: A-Con
B-Con
C-Con

EX:

```
1)  class A {
2)      A() {
3)              System.out.println("A-con");
4)      }
5)  }
6)  class B extends A {
7)  }
8)  class C extends B {
9)      C() {
10)             System.out.println("C-con");
11)     }
12) }
13) class Test {
14)     public static void main(String[] args) {
15)             C c = new C();
16)     }
17) }
```

OUTPUT: A-Con
C-Con

EX:

```
1)  class A {
2)      A(int i) {
3)                  System.out.println("A-int-param-con");
4)      }
5)  }
6)  class B extends A {
7)      B(int i) {
8)                  System.out.println("B-int-param-con");
9)      }
10) }
11) class C extends B {
12)     C(int i) {
13)                 System.out.println("C-int-param-con");
14)     }
15) }
16) class Test {
17)     public static void main(String[] args) {
18)         C c = new C(10);
19)     }
20) }
```

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.

EX:

```
1)  class A {
2)      A() {
3)          System.out.println("A-con");
4)      }
5)      int i = m1();
6)      int m1() {
7)              System.out.println("m1-A");
8)              return 10;
9)      }
```

```
10)      {
11)           System.out.println("IB-A");
12)      }
13) }
14) class B extends A {
15)      int j = m2();
16)      int m2() {
17)                System.out.println("m2-B");
18)                return 20;
19)      }
20)      {
21)           System.out.println("IB-B");
22)      }
23)      B() {
24)           System.out.println("B-Con");
25)      }
26) }
27) class C extends B {
28)      C() {
29)           System.out.println("C-con");
30)      }
31)      {
32)          System.out.println("IB-C");
33)      }
34)      int k = m3();
35)      int m3() {
36)                System.out.println("m3-C");
37)                return 30;
38)      }
39) }
40) class Test {
41)      public static void main(String args[]) {
42)           C c = new C();
43)      }
44) }
```

OUTPUT:

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

EX:

```
1)  class A {
2)      A() {
3)        System.out.println("A-con");
4)      }
5)      static {
6)         System.out.println("SB-A");
7)      }
8)      int m1() {
9)          System.out.println("m1-A");
10)         return 10;
11)     }
12)     static int m2() {
13)        System.out.println("m2-A");
14)        return 20;
15)     }
16)     {
17)        System.out.println("IB-A");
18)     }
19)     static int i=m2();
20)     int j=m1();
21) }
22) class B extends A {
23)     {
24)        System.out.println("IB-B");
25)     }
26)     int m3(){
27)       System.out.println("m3-B");
28)       return 30;
29)     }
30)     static {
31)       System.out.println("SB-B");
32)     }
33)     int k=m3();
34)     B() {
35)      System.out.println("B-Con");
36)     }
37)     static int l=m4();
38)     static int m4(){
39)     System.out.println("m4-B");
40)     return 40;
41) }
42) class C extends B {
43)     static int m5() {
44)        System.out.println("m5-C");
```

```
45)         return 50;
46)     }
47)     int m6() {
48)         System.out.println("m6-C");
49)         return 60;
50)     }
51)     C() {
52)         System.out.println("C-con");
53)     }
54)     int m6();
55)     static int n=m5(); {
56)       System.out.println("IB-C");
57)     }
58)     static {
59)         System.out.println("SB-C");
60)     }
61) }
62) class Test {
63)     public static void main(String args[]) {
64)         C c1=new C();
65)         C c2=new C();
66)     }
67) }
```

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.

EX:

```
1)  class A {
2)     int i=100;
3)     int j=200;
4)  }
5)  class B extends A {
6)     int i=10;
7)     int j=20;
8)     B(int i, int j) {
9)        System.out.println(i+"     "+j);
10)       System.out.println(this.i+"    "+this.j);
11)       System.out.println(super.i+"    "+super.i);
12)    }
13) }
14) class Test {
15)    public static void main(String args[]) {
16)       B b = new B(50,60);
17)    }
18) }
```

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 parameterized constructor at super class, for this, we have to use 'super' keyword.

EX:

```
1)  class A {
2)     A() {
3)       System.out.println("A-Con");
4)     }
5)     A(int i) {
```

```
6)        System.out.println("A-int-param-Con");
7)      }
8)   }
9)   class B extends A {
10)     B() {
11)       super(10);
12)       System.out.println("B-Con");
13)     }
14) }
15) class Test {
16)     public static void main(String args[]) {
17)         B b = new B();
18)     }
19) }
```

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, first, JVM will execute super class 0-arg constructor then JVM will execute 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 go to each and every class available in the source file and checks whether any requirement to provide "default constructors".

b)If any class is identified without user defined constructor explicitly then compiler will add a 0-argument constructor as default constructor.

c)After providing default constructors, compiler will go to 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.
Write the translated code:

```
1)  class A {
2)      A() {
3)              System.out.println("A-con");
4)      }
5)  }
6)  class B extends A {
7)      B(int i) {
8)              System.out.println("B-int-param-con");
9)      }
10) }
11) class C extends B {
12)     C(int i) {
13)             System.out.println("C-int-param-con");
14)     }
15) }
16) class Test {
17)     public static void main(String args[]) {
18)             C c = new C();
19)     }
20) }
```

Status: Compilation Error

EX:

```
1)  class A {
2)      A() {
3)              System.out.println("A-con");
4)      }
5)  }
6)  class B extends A {
7)      B(int i) {
8)              System.out.println("B-int-param-con");
9)      }
10) }
11) class C extends B {
12)     C(int i) {
13)             super(10);
```

```
14)                     System.out.println("C-int-param-con");
15)         }
16) }
17) class Test {
18)         public static void main(String args[]) {
19)                 C c = new C(10);
20)         }
21) }
```

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.

EX:

```
1)  class A {
2)      void m1() {
3)          System.out.println("m1-A");
4)      }
5)  }
6)  class B extends A {
7)      void m2() {
8)          System.out.println("m2-B");
9)          m1();
10)         this.m1();
11)         super.m1();
12)     }
13) }
14) void m1() {
15)     System.out.println("m1-B");
16) }
17) class Test {
18)     public static void main(String args[]) {
19)         B b=new B();
```

```
20)         b.m2();
21)     }
22) }
```

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:

```
1) class A {
2) }
3) class B extends A {
4) }
5) B b = new B();
6) 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.

```
1)  class A {
2)    void m1() {
3)      System.out.println("m1-A");
4)    }
5)  }
6)  class B extends A {
7)     void m2() {
8)       System.out.println("m2-B");
9)     }
10) }
11) class Test {
12)    public static void main(String[] args) {
13)       B b=new B();
14)       b.m1();
15)       b.m2();
16)       A a=b;
17)       a.m1();
18)       //a.m2();---->error
19)    }
20) }
```

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.

EX:

```
1)  class A {
2)       void m1() {
3)            System.out.println("m1-A");
4)       }
5)  }
6)  class B extends A {
```

```
7)        void m2() {
8)            System.out.println("m2-B");
9)        }
10) }
11) class Test {
12)       public static void main(String args[]) {
13)       case 1:
14)               A a = new A();
15)               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 super class 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();
    }
}
```

EX:

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

```
class C extends B {
}
class D extends C {
}
```

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

USES-A Relationship:

This is a relationship between entities, where one entity will use another entity up to a particular action or behavior or method.

To provide USES-A Relationship in java applications, we have to provide 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.

EX:

```
1)  class Account {
2)          String accno;
3)          String accName;
4)          String accType;
5)          int bal = 10000;
6)          Account(String accNo, String accName, String accType) {
7)                  this.accNo = accNo;
8)                  this.accName = accName;
9)                  this.accType = accType;
10)         }
11) }
12) class Transaction {
13)         String tx_tid;
14)         String tx_Type;
15)         Transaction(String tx_id, String tx_Type) {
16)                 this.tx_Id = tx_Id;
17)                 this.tx_Type = tx_Type;
18)         }
19)         public void deposit(Account acc, int dep_Amt) {
20)                 int initial_Amt = acc.bal;
21)                 int total_Avl_Amt = initial_Amt+dep_Amt;
22)                 acc.bal = total_Avl_Amt;
```

```
23)                 System.out.println("Transaction Details");
24)                 System.out.println("--------------------");
25)                 System.out.println("Transaction Id  :"+tx_Id);
26)                 System.out.println("Account Number :"+acc.accNo);
27)                 System.out.println("Account Type  :"+acc.accType);
28)                 System.out.println("Initial Amount :"+initial_Amt);
29)                 System.out.println("Deposit Amount :"+dep_Amt);
30)                 System.out.println("Total Avl Amount :"+total_Avl_Amt);
31)                 System.out.println("Transaction Status:SUCCESS");
32)                 System.out.println("********THANKQ,VISIT AGAIN***********");
33)         }
34) }
35) class UsesAEx {
36)         public static void main(String[] args) {
37)                     Account acc = new Account("abc123","Durga","Savings");
38)                     Transaction tx = new Transaction("T-111","Deposit");
39)                     tx.deposit(acc,5000);
40)         }
41) }
```

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
```

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.
- The main advantage of Polymorphism is "Flexibility" to design Applicatins.

There are 2 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.

EX:

```
1)  class A {
2)      void add(int i,int j) {
3)          System.out.println(i+j);
4)      }
5)      void add(float f1,float f2) {
6)          System.out.println(f1+f2);
7)      }
8)      void add(String str1,String str2) {
9)          System.out.println(str1+str2);
10)     }
11) }
12) class Test {
13)       public static void main(String[] args) {
14)             A a=new A();
15)             a.add(10,20);
16)             a.add(22.22f,33.33f);
17)             a.add("abc","def");
18)       }
19) }
```

OUTPUT:
30
55.550003

abcdef

EX:

```
1)  class Employee {
2)          void gen_Salary(int basic, float hk, float pf, int ta) {
3)               float salary = basic+((basic*hk)/100)-((basic*pf)/100)+ta);
4)               System.out.println("Salary :"+salary);
5)          }
6)          void gen_Salary(int basic, float hk, float pf, int ta, int bonus) {
7)                float salary = basic+((basic*hk)/100)-((basic*pf)/100)+ta)+bonus;
8)                System.out.println("Salary :"+salary);
9)          }
10) }
11) class Test {
12)          public static void main(String[] args) {
13)                  Employee e = new Employee();
14)                  e.gen_Salary(20000,25.of,12.0f,2000);
15)                  e.gen_Salary(20000,25.of,12.0f,2000,5000);
16)          }
17) }
```

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 which we want to override.
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.

EX:

```
1)  class Loan {
2)        public float getIR() {
3)            return 7.0f;
4)        }
5)  }
6)  class GoldLoan extends Loan {
7)          public float getIR() {
8)              return 10.5f;
9)          }
10) }
11) class StudyLoan extends Loan {
12)          public float getIR() {
13)              return 12.0f;
14)          }
15) }
16) class CraftLoan extends Loan {
17) }
18) class Test {
19)        public static void main(String[] args) {
20)                Loan gold_Loan=new GoldLoan();
21)                System.out.println("Gold Loan IR  :"+gold_Loan.getIR()+"%");
22)                Loan study_Loan=new StudyLoan();
23)                System.out.println("Study Loan IR :"+study_Loan.getIR()+"%");
24)                Loan craft_Loan=new CraftLoan();
25)                System.out.println("Craft Loan IR  :"+craft_Loan.getIR()+"%");
26)        }
27) }
```

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.

```
1)  class A {
2)     void m1() {
3)        System.out.println("m1-A");
4)     }
5)  }
6)  class B extends A {
7)       void m1() {
8)          System.out.println("m1-B");
9)       }
10) }
```

```
11) class Test {
12)     public static void main(String args[]) {
13)           /* A a=new A();
14)              a.m1();
15)     Status:Here Method Overriding is not happend, because Mehtod overriding
16)     must require sub class object, not super class object.
17)           */
18)           /*
19)              B b = new B();
20)              b.m1();
21)              Status:Here method Overriding is happened,but,to prove method
        overriding we must require super class reference variable,not sub class reference
        variable,because,subclass reference variable is able to access only sub class
        method when we have the same method in both sub class and super class.
22)           */
23)           A a = new B();
24)           a.m1();
25)     }
26) }
```

Rules to perform Method Overriding:

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

EX:

```
1)  class A {
2)     private void m1() {
3)        System.out.println("m1-A");
4)     }
5)  }
6)  class B extends A {
7)     void m1() {
8)        System.ou.println("m1-B");
9)     }
10) }
11) class Test {
12)    public static void main(String args[]) {
13)       A a = new A();
14)       a.m1();
15)    }
16) }
```

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

EX:

```
1)  class A {
2)    int m1() {
3)      System.ou.println("m1-A");
4)      return 10;
5)    }
6)  }
7)  class B extends A {
8)     void m1() {
9)       System.out.println("m1-B");
10)    }
11) }
12) class Test {
13)     public static void main(String args[]) {
14)          A a=new B();
15)          a.m1();
16)     }
17) }
```

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:

```
1)  class A {
2)      void m1() {
3)           System.out.println("m1-A");
4)      }
5)  }
6)  class B extends A {
7)       final void m1() {
8)            System.out.println("m1-B");
9)       }
10) }
11) class Test {
12)     public static void main(String[] args) {
13)          A a = new B();
14)          a.m1();
15)     }
16) }
```

4.To override super class 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 super class 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.

EX:

```
1)  class A {
2)    static void m1() {
3)       System.out.println("m1-A");
4)    }
5)  }
6)  class B extends A {
7)     static void m1() {
8)        System.out.println("m1-B");
9)     }
10) }
11) class Test {
12)    public static void main(String args[]) {
13)       A a = new B();
14)       a.m1();
15)    }
16) }
```

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:

```
1)  class A {
2)    protected void m1() {
3)       System.out.println("m1-A");
4)    }
5)  }
6)  class B extends A {
7)     public void m1() {
8)       System.out.println("m1-B");
9)     }
10) }
11) class Test {
12)   public static void main(String args[]) {
13)       A a = new A();
```

```
14)         a.m1();
15)     }
16) }
```

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?*

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

EX:

```
1)  class DB_Driver {
2)          public void getDriver() {
3)                  System.out.println("Type-1 Driver");
4)          }
5)  }
6)  class New_DB_Driver extends DB_Driver {
7)            public void getDriver() {
8)                    System.out.println("Type-4-Driver");
9)            }
10) }
11) class Test {
12)        public static void main(String args[]) {
13)                DB_Driver driver = new New_DB_Driver();
14)                driver.getDriver();
15)        }
16) }
```

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.

```
1)  abstract class DB_Driver {
2)        public abstract void getDriver();
3)  }
4)  class New_DB_Driver extends DB_Driver {
5)          public void getDriver() {
6)                  System.out.println("Type-4 Driver");
7)          }
8)  }
9)  class Test {
10)      public static void main(String args[]) {
11)            DB_Driver driver = new New_DB_Driver();
12)            driver.getDriver();
13)      }
14) }
```

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.

```
1)  abstract class A {
2)     void m1() {
3)        System.out.println("m1-A");
4)     }
5)     abstract void m2();
6)     abstract void m3();
7)  }
8)  class B extends A {
9)     void m2() {
10)      System.out.println("m2-B");
11)    }
12)    void m3() {
13)       System.out.println("m3-B");
14)    }
15)    void m4() {
```

```
16)       System.out.println("m4-B");
17)    }
18) }
19) class Test {
20)    public static void main(String args[]) {
21)       A a = new B();
22)       a.m1();
23)       a.m2();
24)       a.m3();
25)       //a.m4();---error
26)       B b = new B();
27)       b.m1();
28)       b.m2();
29)       b.m3();
30)       b.m4();
31)    }
32) }
```

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.

EX:

```
1)  abstract class A {
2)     A() {
3)       System.out.println("A-Con");
4)     }
5)  }
6)  class B extends A {
7)     B() {
8)       System.out.println("B-Con");
9)     }
10) }
11) class Test {
12)    public static void main(String[] args) {
13)       B b = new B();
14)    }
15) }
```

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.

EX:

```
1)  abstract class A {
2)      abstract void m1();
3)      abstract void m2();
4)      abstract void m3();
5)  }
6)  abstract class B extends A {
7)      void m1() {
8)        System.out.println("m1-A");
9)      }
10) }
11) class C extends B {
12)    void m2() {
13)      System.out.println("m2-C");
14)    }
15)    void m3() {
16)      System.out.println("m3-C");
17)    }
18) }
19) class Test {
20)     public static void main(String[] args) {
21)         A a = new C();
22)         a.m1();
23)         a.m2();
24)         a.m3();
25)     }
26) }
```

In Java applications, if we want to declare an abstract class then it is not at all mandatory condition to have at least 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.

```
1)  abstract class A {
2)      void m1() {
3)         System.out.println("m1-A");
4)      }
5)  }
6)  class B extends A {
7)      void m2() {
8)        System.out.println("m2-B");
9)      }
10) }
11) class Test {
12)    public static void main(String args[]) {
```

```
13)       A a = new B();
14)       a.m1();
15)       //a.m2();---------->Error
16)       B b = new B();
17)       b.m1();
18)       b.m2();
19)    }
20) }
```

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

```
1)  class A {
2)     void m1() {
3)          System.out.println("m1-A");
4)     }
5)  }
6)  abstract class B extends A {
7)       abstract void m2();
8)  }
9)  class C extends B {
10)       void m2() {
11)          System.out.println("m2-C");
12)      }
13)      void m3() {
14)           System.out.println("m3-C");
15)      }
16) }
17) class Test {
18)     public static void main(String args[]) {
19)           A a = new C();
20)           a.m1();
21)           //a.m2();---◊error
22)           //a.m3();---◊error
23)           B b = new C();
24)           b.m1();
25)           b.m2();
26)           //b.m3();---◊error
27)           C c = new C();
28)           c.m1();
29)           c.m2();
30)           c.m3();
31)     }
32) }
```

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.

```
1)  interface I {
2)      void m1();
3)      void m2();
4)      void m3();
5)  }
6)  class A implements I {
7)     public void m1() {
8)         System.out.println("m1-A");
9)     }
10)    public void m2() {
```

```
11)         System.out.println("m2-A");
12)     }
13)     public void m3() {
14)         System.out.println("m3-A");
15)     }
16)     public void m4() {
17)       System.out.println("m4-A");
18)     }
19) }
20) class Test {
21)     public static void main(String args[]) {
22)         I i = new A();
23)         i.m1();
24)         i.m2();
25)         i.m3();
26)         //i.m4();------>error
27)         A a = new A();
28)         a.m1();
29)         a.m2();
30)         a.m3();
31)         a.m4();
32)     }
33) }
```

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.

```
1)  interface I {
2)    void m1();
3)    void m2();
4)    void m3();
5)  }
6)  abstract class A implements I {
7)      public void m1() {
8)         System.out.println("m1-A");
9)      }
10) }
11) class B extends A {
12)    public void m2() {
13)        System.out.println("m2-B");
```

```
14)   }
15)   public void m3() {
16)     System.out.println("m3-B");
17)   }
18) }
19) class Test {
20)   public static void main(String args[]) {
21)     I i = new I();
22)     i.m1();
23)     i.m2();
24)     i.m3();
25)     A a = new B();
26)     a.m1();
27)     a.m2();
28)     a.m3();
29)     B b = new B();
30)     b.m1();
31)     b.m2();
32)     b.m3();
33)   }
34) }
```

Difference between classes , abstract classes and interface

1) classes - concreate methods only
abstract classes - both
interface - only abstract methods

2) classes - use class keyword
abstract - use abstract keyword with class
interface - use interface keyword

3) classes - both Object and reference variable
abstract class and interface only reference variable

4) classes - no default cases for variables are available for classes and abstract classes
interface - bydefault public static final

5) classes - no default cases for methods are available for classes and abstract classes
interface - bydefault public abstract

6) classes - no default cases for inner classes in classes and abstract classes
interface - bydefault inner classes are static in interface

Type text here

7) classes and abstract classes - constructor are allowed
interface - constructors are not allowed

8) classes and abstract classes - static block , instance block are allowed
interface - no static block , instance block are allowed

9) classes and abstract classes - static methods , instance methods are allowed
interface - no static methods , instance methods are allowed

10) classes - less sharability
abstract classes - middle sharability
interface - more sharability

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.

EX:

```
1)  interface I1{
2)    void m1();
3)  }
4)  interface I2 {
5)    void m2();
6)  }
7)  interface I3 extends I1, I2 {
8)    void m3();
9)  }
10) class A implements I3 {
11)   public void m1() {
12)     System.out.println("m1-A");
13)   }
14)   public void m2() {
15)     System.out.println("m2-A");
16)   }
17)   public void m3() {
18)     System.out.println("m3-A");
19)   }
```

```
20) }
21) class Test {
22)     public static void main(String args[]) {
23)         I1 i1 = new A();
24)         i1.m1();
25)         I2 i2 = new A();
26)         i2.m2();
27)         I3 i3 = new A();
28)         i3.m1();
29)         i3.m2();
30)         i3.m3();
31)         A a = new A();
32)         a.m1();
33)         a.m2();
34)         a.m3();
35)     }
36) }
```

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

```
1)  interface I1 {
2)     void m1();
3)  }
4)  interface I2 {
5)     void m2();
6)  }
7)  interface I3 {
8)     void m3();
9)  }
10) class A implements I1, I2, I3 {
11)     public void m1() {
12)         System.out.println("m1-A");
13)     }
14)     public void m2() {
15)        System.out.println("m2-A");
16)     }
17)     public void m3() {
18)         System.out.println("m3-A");
19)     }
20) }
21) class Test {
22)     public static void main(String args[]) {
23)         I1 i1 = new A();
24)         i1.m1();
```

```
25)     I2 i2 = new A();
26)     i2.m2();
27)     I3 i3=new A();
28)     i3.m3();
29)     A a = new A();
30)     a.m1();
31)     a.m2();
32)     a.m3();
33)   }
34) }
```

Q) Fond Valid Syntaxes between classes, abstract classes and Interfaces from the following list of syntaxes?

1) class extends class --->Valid
2) class extends class,class--->InValid
3) class extends abstract class--->Valid
4) class extends abstract class,class--->InValid
5) class extends abstract class,abstract class--->InValid
6) class extends interface--->InValid
7) class implements interface--->Valid
8) class implements interface,interface--->Valid
9) class implements interface extends class--->InValid
10) class implements interface extends abstract class--->InValid
11) class extends class implements interface--->Valid
12) class extends abstract class implements interface--->Valid
13) abstract class extends class--->Valid
14) abstract class extends abstract class--->Valid
15) abstract class extends class,class--->InValid
16) abstract class extends abstract class,abstract class--->InValid
17) abstract class extends class,abstract class--->InValid
18) abstract class extends interface--->InValid
19) abstract class implements interface--->Valid
20) abstract class implements interface,interface--->Valid
21) abstract class extends class implements interface--->Valid
22) abstract class extends abstract class implements interface--->Valid
23) abstract class implements interface extends class-->InValid
24) abstract class implements interface extends abstract class-->InValid
25) interface extends interface -->Valid
26) interface extends interface,interface -->Valid
27) interface extends class -->InValid
28) interface extends abstract class -->InValid
29) interface implements interface -->InValid

Marker Interfaces:

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 , java.lang.Cloneable

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.

java.lang.Cloneable:

The process of generating duplicate object is called as Object Cloning.
IN java applications, by default, all objects are not eligible for Object cloning, only the objects which implements java.lang.Cloneble interface are eligible for Object cloning.

JAVA 8 Features over Interfaces

- Default Methods in Interfaces
- Static Methods in Interfaces
- Functional Interfaces

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 number 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:

```
1) interface I {
2)    default void m1() {
3)       System.out.println("m1-A");
4)    }
5) }
6) class A implements I {}
7) class Test {
8)    public static void main(String args[]) {
9)       I i = new A();
```

```
10)       i.m1();
11)   }
12) }
```

NOTE: It is possible to provide more than one default methods within 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.

```
1)  interface I {
2)     default void m1() {
3)        System.out.println("m1-A");
4)     }
5)  }
6)  class A implements I {
7)      public void m1() {
8)        System.out.println("m1-A");
9)      }
10) }
11) class Test {
12)   public static void main(String args[]) {
13)       I i = new A();
14)       i.m1();
15)   }
16) }
```

Static Methods in Interfaces:

Up to 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

EX:

```
1)  interface I {
2)    static void m1() {
3)       System.out.println("m1-1");
4)    }
5)  }
6)  class Test {
7)     public static void main(String args[]) {
8)       I.m1();
9)     }
10) }
```

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

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 number of default methods and any number of static methods.

```
1)  @Functional Interface
2)  interface I {
3)     void m1();
4)     //void m2();---->error
5)     default void m3() {
6)        System.out.println("m3-I");
7)     }
8)     static void m4() {
9)        System.out.println("m4-I");
10)    }
11)    class A implements I {
12)       public void m1() {
13)          System.out.println("m1-A");
```

```
14)     }
15) }
16) class Test {
17)     public static void main(String args[]) {
18)         I i=new A();
19)         i.m1();
20)         i.m3();
21)         //i.m4();--->error
22)         I.m4();
23)         //A.m4();--->error
24)     }
25) }
```

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".

if ref_Var is super class to the specified class name and it contains sub class object reference value then instanceof operator will return true value

EX:

```
1)  class A {
2)  }
3)  class B extends A {
4)  }
5)  class C {
6)  }
7)  class Test {
8)      public static void main(String args[]) {
9)          A a new A();
10)         B b = new B();
11)         System.out.println(a instanceof A);
12)         System.out.println(a instanceof B);
```

```
13)         System.out.println(b instanceof A);
14)         //System.out.println(a instanceof C);
15)     }
16) }
```

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 in order 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.

EX:

```
1)  class Student implements Cloneable {
2)      String sid;
3)      String sname;
4)      String saddr;
5)      Student(String sid, String sname, String saddr) {
6)          this.sid = sid;
7)          this.sname = sname;
8)          this.saddr = saddr;
9)      }
10)     public Object clone() throws CloneNotSupportedException {
11)         return super.clone();
12)     }
13)     public String toString() {
14)         System.out.println("Student details");
15)         System.out.println("----------------");
16)         System.out.println("Student Id :"+sid);
17)         System.out.println("Student name:"+sname);
18)         System.out.println("Student Address:"+saddr);
19)         return "";
20)     }
21) }
22) class Test {
23)     public static void main(String args[]) {
24)         Student std1 = new Student("S-111","Durga","Hyd");
```

```
25)         System.out.println("Student Details Before Cloning");
26)         System.out.println(std1);
27)
28)         Student std2 = (Student)std1.clone();
29)         System.out.println();
30)         System.out.println("Student Details After cloning");
31)         System.out.println(std2);
32)     }
33) }
```

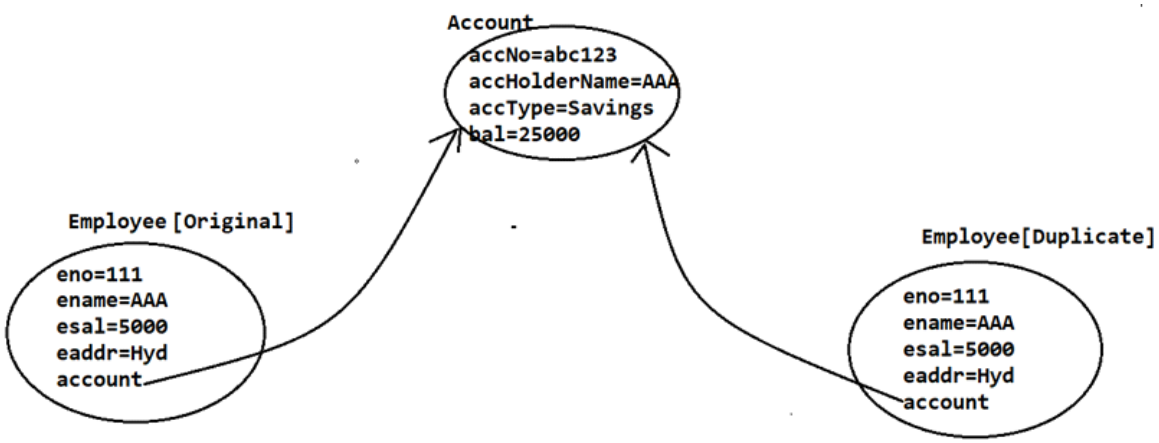
There are 2 types clonings in Java:

- Shallow Cloning/Shallow Copy
- Deep Cloning/Deep Copy

Shallow Cloning/Shallow Copy:

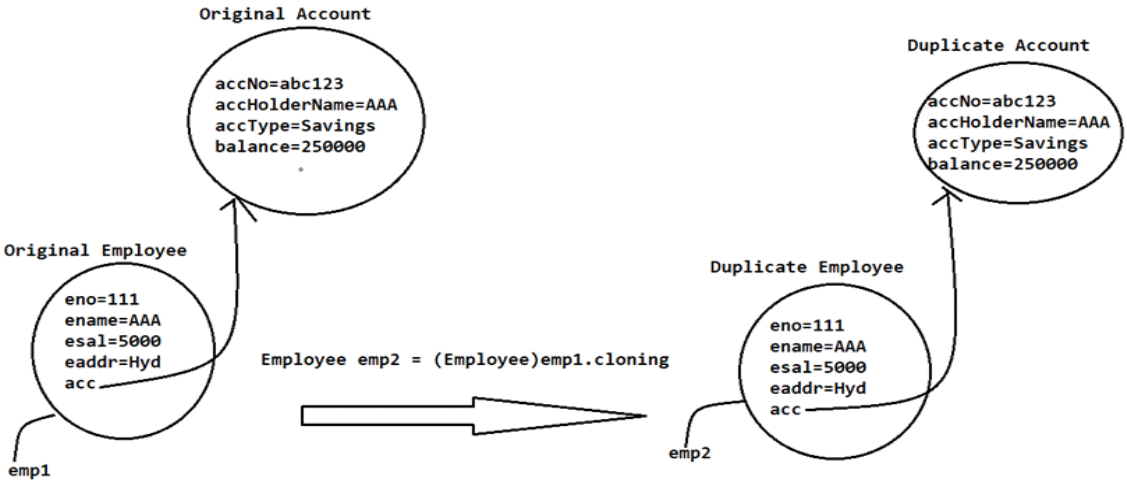
In this cloning mechanism, while cloning an object if any associated object is encountered then JVM will not duplicate associated object along with data duplication, where duplicated object is also refer the same associated object which was refereed by original object.

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.



EX:

```
1)  class Account {
2)        String accNo;
3)        String accName;
4)        String accType;
5)        Account(String accNo, String accName, String accType) {
6)                this.accNo = accNo;
7)                this.accName = accName;
8)                this.accType = accType;
9)        }
10) }
11) class Employee implements Cloneable {
12)        String eid;
13)        String ename;
14)        String eddr;
15)        Account acc;
16)        Employee(String eid, String ename, String eaddr, Account acc) {
17)                this.eid = eid;
18)                this.ename = ename;
19)                this.eaddr = eaddr;
20)                this.acc = acc;
21)        }
22)        public Object clone() throws CloneNotSuppoertException {
23)            return super.clone();
24)        }
25)        public String toString() {
26)                System.out.println("Employee Details");
27)                System.out.println("----------------");
28)                System.out.println("Employee Id  :"+eid);
29)                System.out.println("Employee Address :"+eaddr);
30)                System.out.println("Employee Name   :"+ename);
```

if ref_var is super class to the specified class name and it contains sub class object reference value then instanceof operator

```
31)                 System.out.println("Account Deatails");
32)                 System.out.println("---------------");
33)                 System.out.println("Account Number:"+acc.accNo);
34)                 System.out.println("Account Name :"+acc.accName);
35)                 System.out.println("Account Type :"+acc.accType);
36)                 System.out.println("Account Reference: "+acc);
37)                 return "";
38)         }
39) }
40) class Test {
41)       public static void main(String args[]) throws Exception {
42)                 Account acc = new Account("abc123","Durga","Savings");
43)                 Employee emp1 = new Employee("E-111","Durga","Hyd",acc);
44)                 System.out.println("Employee Details Before Cloning");
45)                 System.out.println(emp1);
46)                 Employee emp2 = (Employee)emp1.clone();
47)                 System.out.println("Employee Details After Cloning");
48)                 System.out.println(emp2);
49)       }
50) }
```

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;
```

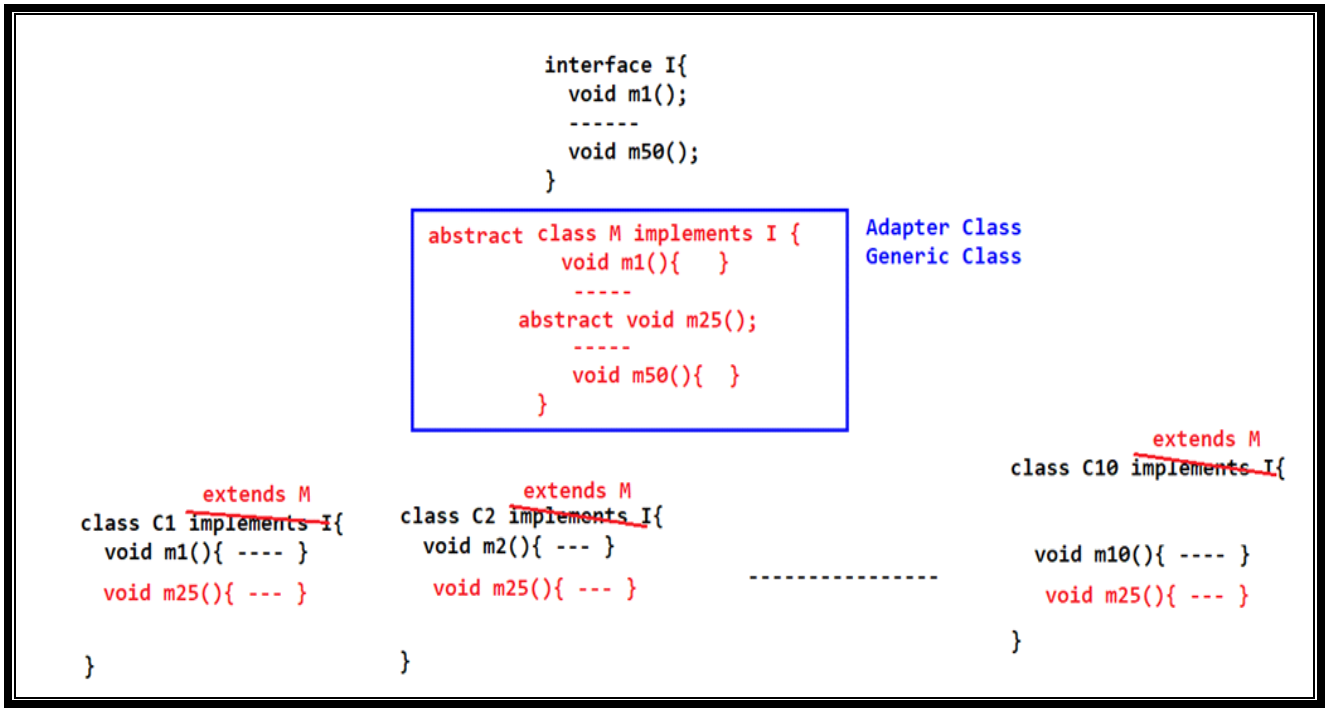
Adapter Classes:

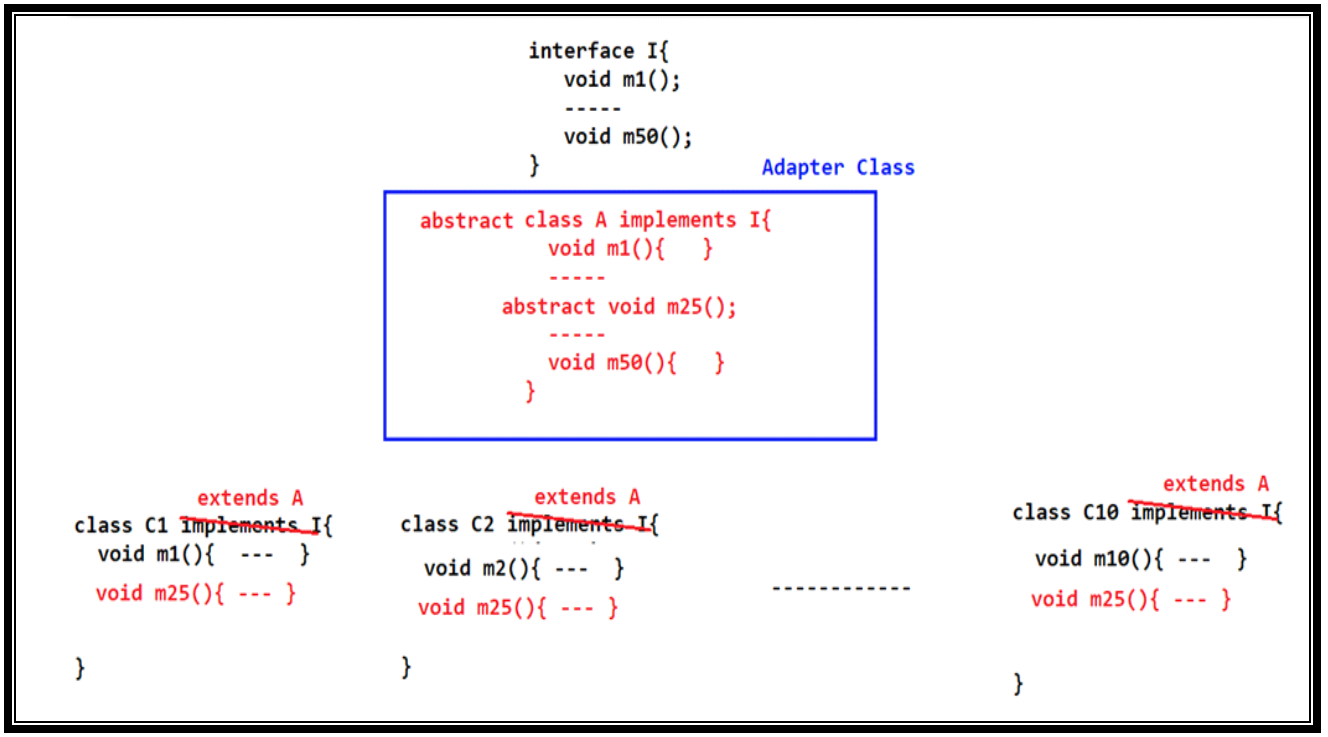
In general, in Java applications when we implements an interface in any class then we must implement all the abstract methods of that interface with or without the requirement, it may increase unnecessary methods implementation in Java applications.

To overcome the above problem we have to use “Adapter classes“.

```
interface I{
  void m1();
  ------
  void m50();
}

abstract class M implements I {
    void m1(){   }
    -----
  abstract void m25();
    -----
    void m50(){  }
 }

Adapter Class
Generic Class

class C1 extends M implements I{
  void m1(){ ---- }
  void m25(){ --- }
}

class C2 extends M implements I{
  void m2(){ --- }
  void m25(){ --- }
}

----------

class C10 extends M implements I{
  void m10(){ ---- }
  void m25(){ --- }
}
```


```
interface I{
  void m1();
  ------
  void m50();
}

abstract class M implements I {
    void m1(){   }
    -----
  abstract void m25();
    -----
    void m50(){  }
 }

Adapter Class
Generic Class

class C1 extends M implements I{
  void m1(){ ---- }
  void m25(){ --- }
}

class C2 extends M implements I{
  void m2(){ --- }
  void m25(){ --- }
}

----------

class C10 extends M implements I{
  void m10(){ ---- }
  void m25(){ --- }
}
```





The main intention of Adapter classes is to reduce unnecessary methods implementations while implementing interfaces in implementation classes.

EX:

WindowAdapter
MouseAdapter
KeyAdapter
GenericServlet
ServletRequestWrapper
ServletResponseWrapper
HttpServletRequestWrapper
HttpServletResponseWrapper