

```

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, by default, enums are final classes.

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

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:

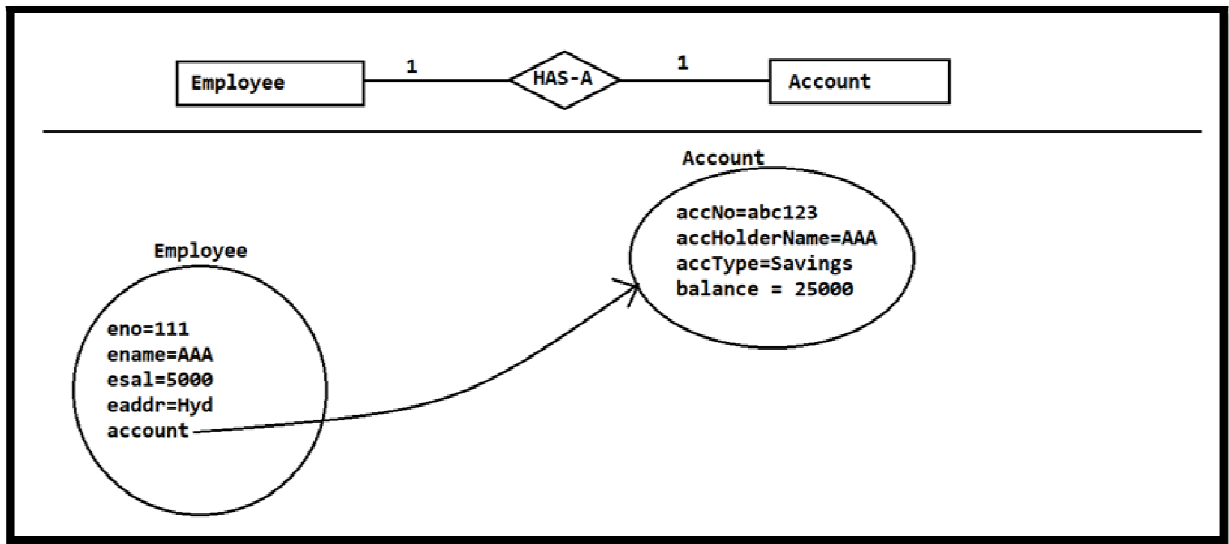
Employee Details

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 accBalance) {
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 dependant 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 setName(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) }

```

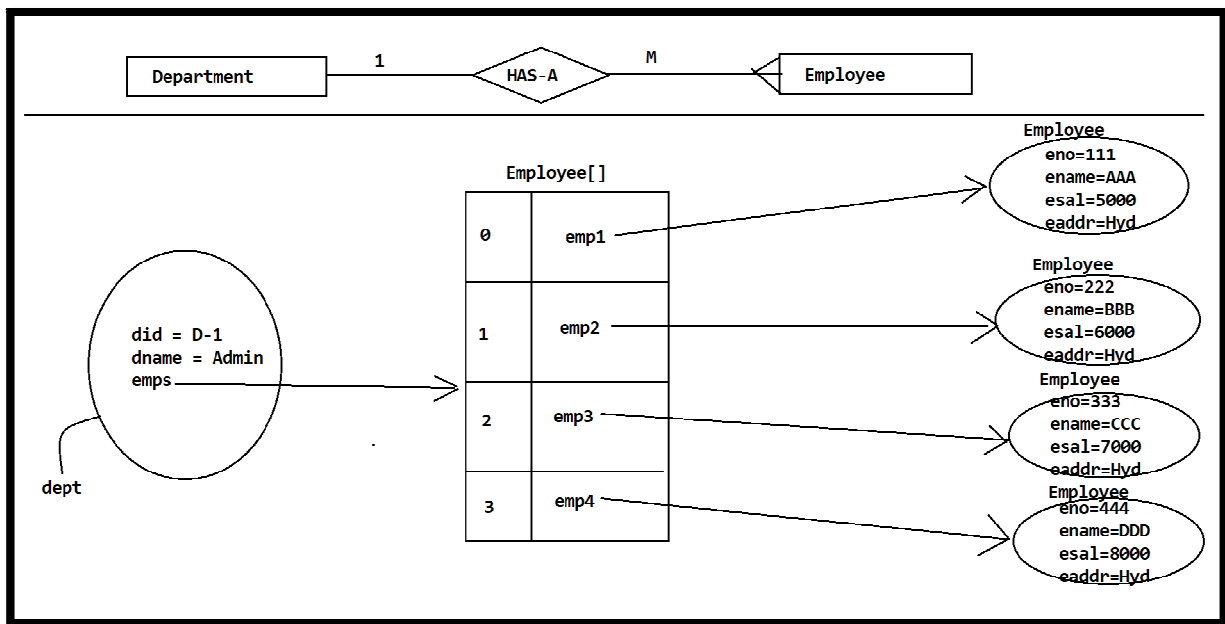
Test.java

```

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) }

```

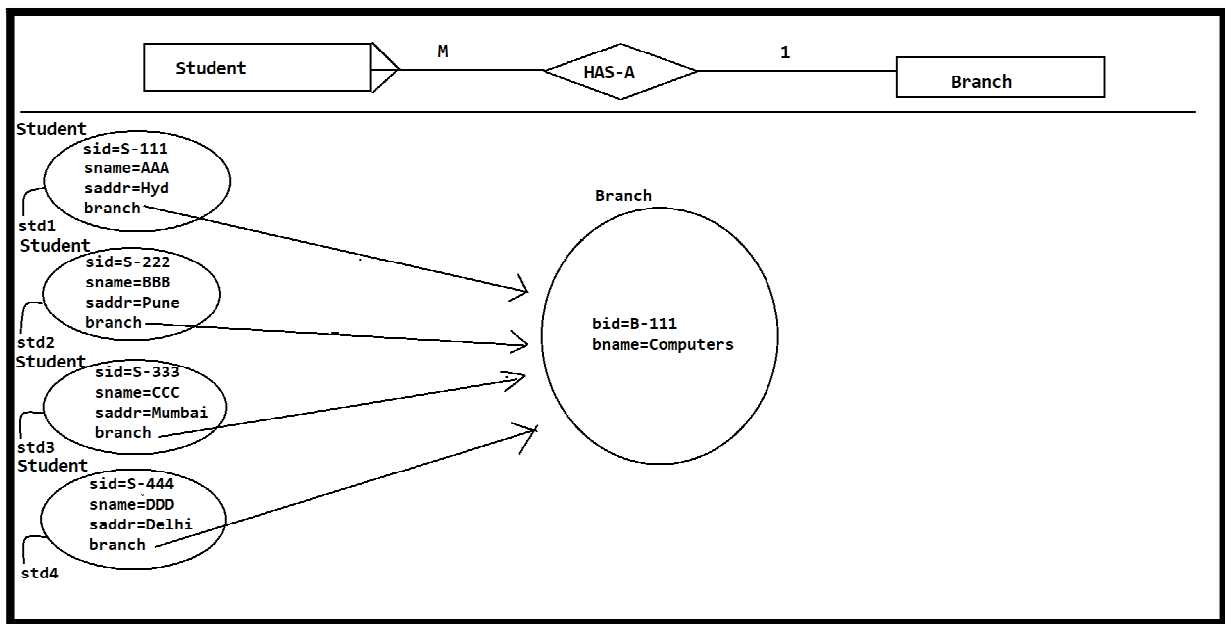
Test.java

```

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 aetCname() {
```

```
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) }

```

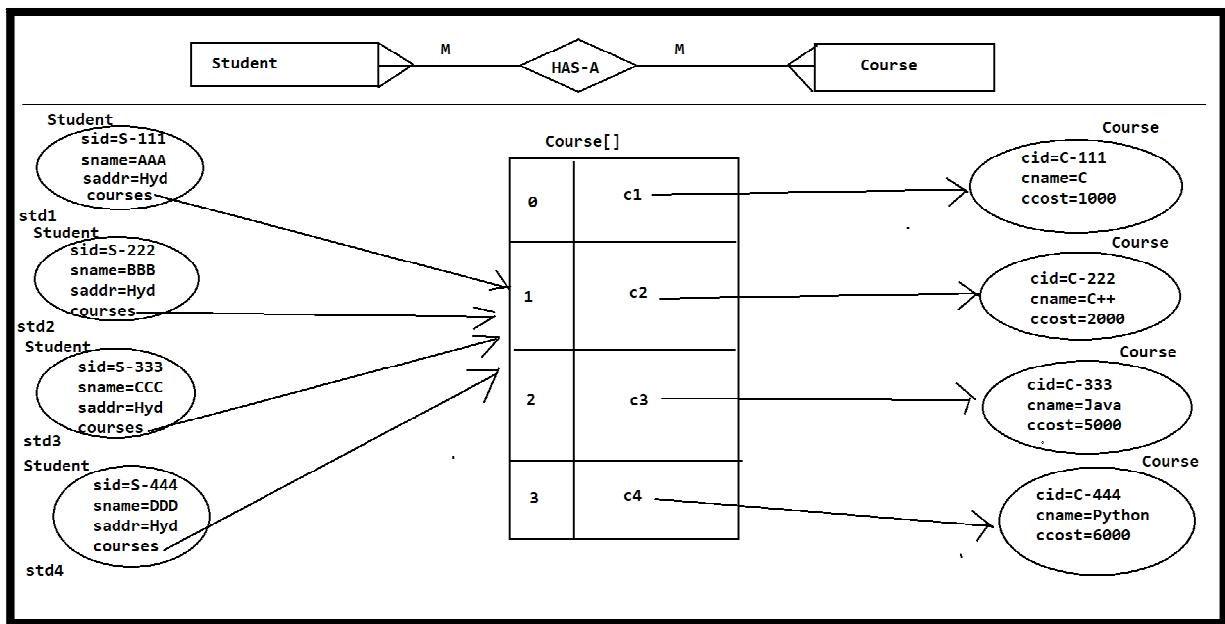
Test.java

```

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.

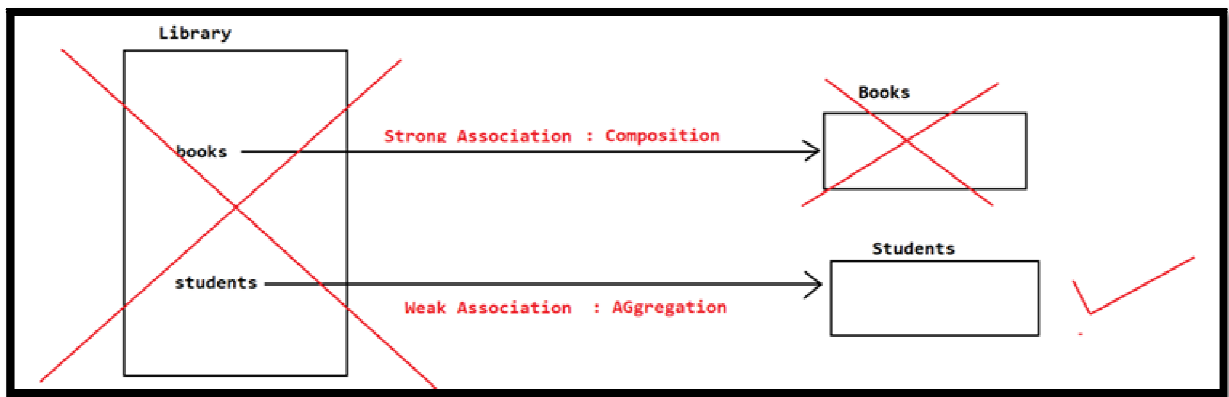
2. In case of Aggregation, if contained object is existed even without container object.
In case of Composition, if contained object is not existed without container object.

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.

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.

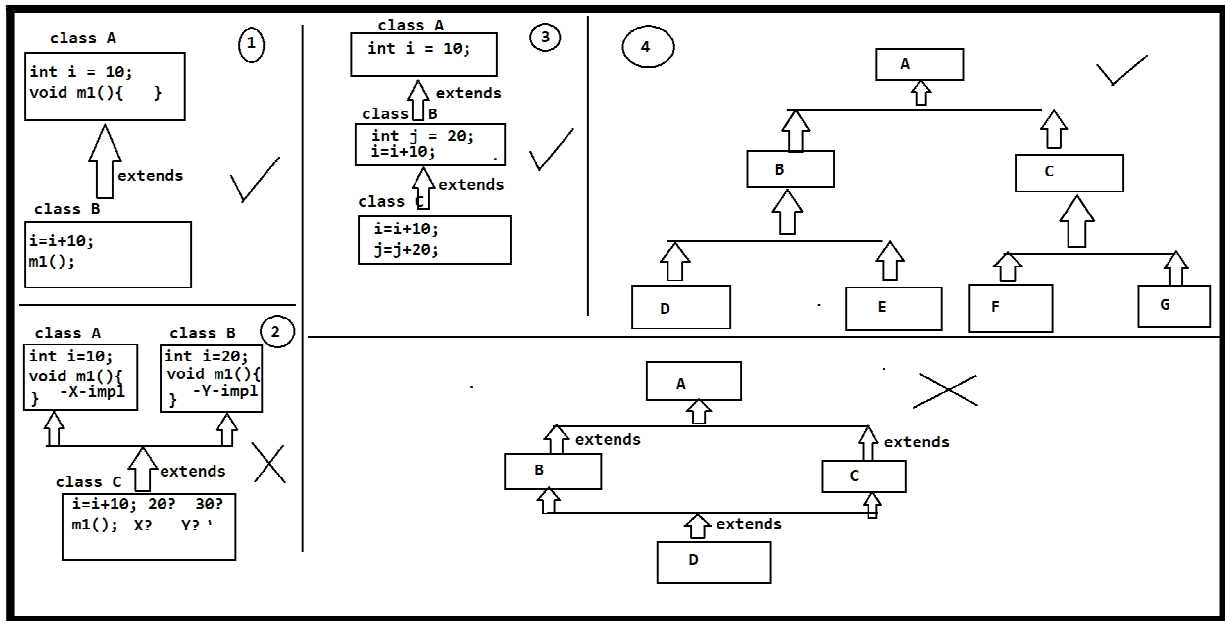
EX:

```
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)         eaddr = 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) }
```

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 without "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(20000,25.0f,12.0f,2000);
15)         e.gen_Salary(20000,25.0f,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 from 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.out.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();
```