Unit IV

4 Inheritance and Polymorphism

4.1 : Introduction and Need for Inheritance

Q.1 What is inheritace ?

☞ [SPPU : June-22, Marks 2]

Ans. : Inheritance is a mechanism in Java by which derived class can borrow the properties of base class and at the same time the derived class may have some additional properties.

Q.2 Explain the concept of base class and derived class.

Ans. :

- The inheritance is a mechanism in which the child class is derived from a parent class.
- This derivation is using the keyword **extends**.
- The parent class is called **base** class and child class is called **derived class**.
- For example

```
Class A                          ← This is Base class
{
    ...
}
Class B extends A                ← This is Derived class
{
    ...                          // uses properties of A
}
```

Q.3 Explain the need for inheritance.

Ans. :

1) It helps in reduced code.
2) It makes use of reusability of code.

3) It enhances readability of code.
4) Execution of code is efficient.

4.2 : Types of Inheritance

Q.4 What are various types of inheritance ?

☞ [SPPU : June-22, Marks 2]

Ans. : 1. Single inheritance :

- In single inheritance there is one parent per derived class. This is the most common form of inheritance.

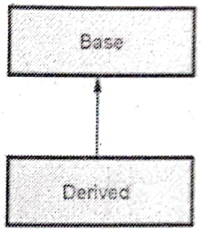


Fig. Q.4.1 Single inheritance

2. Multiple inheritance :

- In multiple inheritance the derived class is derived from more than one base class.

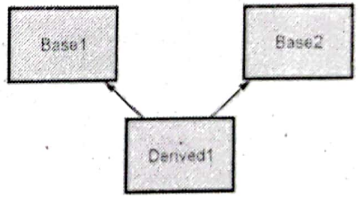


Fig. Q.4.2 Multiple inheritance

- Java **does not implement multiple inheritance** directly but it makes use of the concept called interfaces to implement the multiple inheritance.

3. Multilevel inheritance :

- When a derived class is derived from a base class which itself is a derived class then that type of inheritance is called multilevel inheritance.
- For example - If class A is a base class and class B is another class which is derived from A, similarly there is another class C being derived from class B then such a derivation leads to multilevel inheritance.

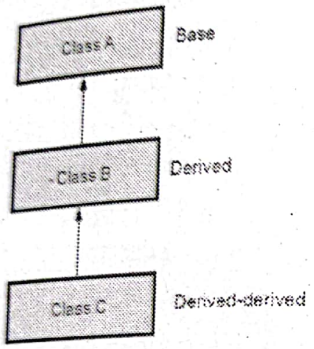


Fig. Q.4.3 Multilevel inheritance

4. Hybrid inheritance :

- When two or more types of inheritances are combined together then it forms the hybrid inheritance. The following Fig. Q.4.3 represents the typical scenario of hybrid inheritance.

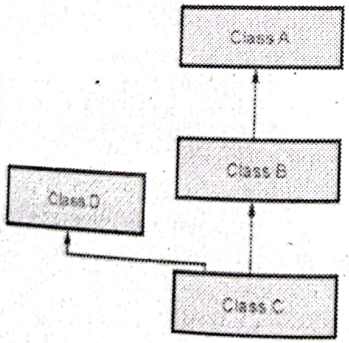


Fig. Q.4.4 Hybrid inheritance

4.3 : Implementation

Q.5 Write a Java program to implement single level inheritance.
OR How can you inherit a class in Java ?

☞ **[SPPU : June-22, Marks 5]**

Ans. :

```
class A
{
        int a;
        void set_a(int i)
        {
                a=i;
        }
        void show_a()
        {
                System.out.println("The value of a= "+a);
        }
}
class B extends A  //extending  the base class A
{
        int b;
        void set_b(int i)
        {
                b=i;
        }
        void show_b()
        {
                System.out.println("The value of b= "+b);
        }
        void mul()
        {
                int c;
                c=a*b;
                System.out.println(" The value of c= "+c);
        }
}
class InheritDemo1
{
        public static void main(String args[])
        {
                A obj_A=new A();
                B obj_B=new B();
                obj_B.set_a(10);
```

Note that object of class B is accessing method of class A

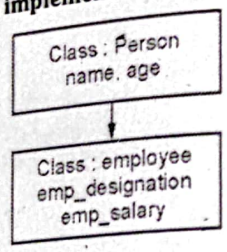
```
        obj_B.set_b(20);//10
        obj_B.show_a();//20
        obj_B.show_b();//200
        obj_B.mul();

    }
}
```

Output

```
The value of a= 10
The value of b= 20
The value of c= 200
```

Q.6 Write a program to implement following inheritance :



Ans. :

```
class Person
{
    String name;
    int age;
    void get_personInfo(int ag,String nm)
    {
        age=ag;
        name=nm;
    }
    void display_personInfo()
    {
        System.out.println("Name: "+name);
        System.out.println("age: "+age);
    }
}
class Employee extends Person
    //extending  the base class Person
{
    String emp_designation;
    float emp_salary;
```



```
    void get_employeeInfo(String designation,float salary)
    {
        emp_designation=designation;
        emp_salary=salary;
    }
    void display_empInfo()


    {
        System.out.println("Employee
        Designation:"+emp_designation);
        System.out.println("Employee Salary: "+emp_salary);
    }
}
class InheritDemo
{
    public static void main(String args[])
    {
        Employee e=new Employee();
        e.get_personInfo(25,"Ankita");
        e.display_personInfo();
        e.get_employeeInfo("Manager",10000);
        e.display_empInfo();
    }
}
```

Output

```
Name: Ankita
age: 27
Employee Designation: Manager
Employee Salary: 10000.0
```

Q.7 Write a Java code to implement multilevel inheritance

Ans. : Java Program[MultiLvlInherit.java]

```
class A
{
 int a;
 void set_a(int i)
 {
   a=i;
 }
```

```
void show_a()
{
System.out.println("The value of a= "+a);
}
}
class B extends A
{
int b;
void set_b(int i)
{
 b=i;
}
void show_b()
{
System.out.println("The value of b= "+b);
}
}
class C extends B
{
int c;
void set_c(int i)
{
 c=i;
}
void show_c()
{
System.out.println("The value of c= "+c);
}
void mul()
{
int ans;
ans=a*b*c;
System.out.println(" The value of ans= "+ans);
}
}
class MultiLvlInherit
{
public static void main(String args[])
{
A obj_A=new A();
B obj_B=new B();
```



```
C obj_C=new C();
obj_C.set_a(10);
obj_C.set_b(20);
obj_C.set_c(30);
obj_C.show_a();//10
obj_C.show_b();//20
obj_C.show_c();//30
obj_C.mul();//6000
}
}
```

Output

```
The value of a= 10
The value of b= 20
The value of c= 30
The value of ans= 6000
```

4.4 : Benefits and Cost of Inheritance

Q.8 What are the benefits of inheritance ?
Ans. :

1. **Software reusability :** The child class inherits some behaviour of the parent class. Hence for the child class it is not necessary to write the code for the inherited behaviour.
2. **Code sharing :** The same class can be used by many applications or users. At other level the two or more classes developed by a single programmer may be inherited from a single parent class.
3. **Software components :** Using inheritance the reusable software components can be created. Due to creation of such software components new or novel applications can be created with actually no coding or little coding.
4. **Consistency of interface :** When various subclasses are derived from the same superclass then all these subclasses will have the same behaviour. Thus the similar objects have the similar interface and due to which the user will be presented with the collection of objects having similar behaviour.

5. **Rapid prototyping :** Rapid prototyping refers to creation of the basic system in efficient and quick manner. Many software components can be inherited from the parent class and interfaces. Due to which very small amount of code needs to be written to build a prototype system.
6. **Polymorphism and frameworks :** Using polymorphism the higher level components can be created by using different lower level parts that for the suitable application.
7. **Information hiding :** The software component can be reused by the programmer without bothering about its implementation details. This reduces the interconnected nature between the several software systems.

Q.9 What is the cost of inheritance ?

Ans. :

1. **Execution speed :** The inherited methods are often slower than the specialized code because the inherited method have to deal with arbitrary number of subclasses and interfaces. But this reduction in speed can be compensated by increase in speed of program development.
2. **Program size :** Due to use of library of software component the program size gets increased heavily. But due to this memory cost gets decreased and high quality, error free code can be produced.
3. **Message passing overhead :** Communication among the objects occurs using by passing the messages. It is observed that the message passing operation is more costly than the procedure invocation. But the message passing mechanism can be tolerated because of the benefits that are getting due to object oriented techniques.

   By eliminating the polymorphism and by using the dynamic methods or virtual methods the overhead of message passing mechanism can be reduced.

4. **Program complexity :** Due to overuse of inheritance the program complexity gets increased. In order to understand the control flow of the program, it is essential to scan the inheritance graph up and down for several times. Reading and scanning the inheritance graph in an up and down manner for understanding the program flow is called the **yo yo problem**.

4.5 : Constructors in Derived Classes

Q.10 Explain the concept of constructor chaining with suitable example.

Ans. : • Normally a superclass's constructor is called before the subclass's constructor. This is called **constructor chaining**.

• Thus the calling of constructor occurs from top to down.

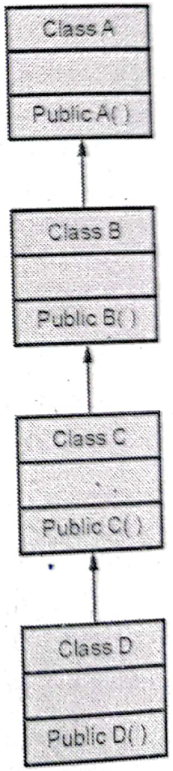


Fig. Q.10.1 Constructor call

- The constructor chaining can be illustrated by following Java program.

Java Program[MainClass.java]

```
class A
{
public A()   //constructor defined
      {
            System.out.println("1. Statement in class A");
      }
}
class B extends A //Derived child of A
{
      public B() //constructor defined
      {
      System.out.println("2. Statement in class B");
      }
}
class C extends B    //derived class of class B
{
      public C()    //constructor defined
      {
            System.out.println("3. Statement in class C");
      }
}
class D extends C  //derived class of class C
{
      public D()  //constructor defined
      {
            System.out.println("4. Statement in class D");
   }
}
class MainClass
{
   public static void main(String args[])
   {
      D obj=new D(); //calling constructor using derived class object
   }
}
```


Output

```
1. Statement in class A
2. Statement in class B
3. Statement in class C
4. Statement in class D
```

4.6 : Method Overriding

Q.11 Explain the concept of method overriding with an example.

Ans. : • Method overriding is a mechanism in which a subclass **inherits the methods** of superclass and sometimes the **subclass modifies** the implementation of a method defined in superclass.

- The method of superclass which gets modified in subclass has the same name and type signature.
- The overridden method must be called from the subclass.
- Consider following Java Program, in which the method(named as **fun**) in which **a** is assigned with some value is modified in the derived class. When an overridden method is called from within a subclass, it will always refer to the version of that method re-defined by the subclass. The version of the method defined by the superclass will be hidden.

Java Program[OverrideDemo.java]

```
class A
{
      int a=0;
      void fun(int i)
      {
            this.a=i;
      }
}
class B extends A
{
      int b;
      void fun(int i)
      {

            int c;
```

```
        b=20;
        super.fun(i+5);
        System.out.println("value of a:"+a);
        System.out.println("value of b:"+b);
        c=a*b;
        System.out.println("The value of c= "+c);
    }
}
class OverrideDemo
{
    public static void main(String args[])
    {
        B obj_B=new B();
        obj_B.fun(10);//function re-defined in derived class
    }
}
```

Output

```
value of a:15
value of b:20
The value of c= 300
```

Q.12 What is the difference between method overloading and method overriding.

Ans. :

Method overloading	Method overriding
The method overloading occurs at compile time.	The method overriding occurs at the run time or **execution time**.
In case of method overloading **different number of parameters** can be passed to the function.	In function overriding the **number of parameters** that are passed to the function are the same.
The overloaded functions may have **different return types**.	In method overriding all the methods will have the **same return type**.
Method overloading is performed within a class.	Method overriding is normally performed between two classes that have inheritance relationship.

4.7 : Abstract Classes

Q.13 What is abstract class ?

Ans. : Abstract class is a class which is declared with abstract keyword. It may contain both abstract and non-abstract methods (no-abstract method means the method with functional body).

For example -

```
abstract class A
{
abstract void fun1();
void fun2()
{
    System.out.println("A:In fun2");
}
}
```

This method is so abstract that it has no definition body.

4.8 : Interfaces

Q.14 What is interfaces ? Also give the difference between class and interface.

☞ **[SPPU : June-22, Marks 2]**

Ans. : An interface is defined as an abstract type used to specify the behavior of a class.

Difference between class and interface :

- An interface is similar to a class but there lies some difference between the two.

Class	Interface
The class is denoted by a keyword **class**.	The interface is denoted by a keyword **interface**.
The class contains data members and methods. But the methods are defined in class implementation. Thus class contains an executable code.	The interfaces may contain data members and methods but the methods are not defined. The interface serves as an outline for the class.

By creating an instance of a class the class members can be accessed.	You can not create an instance of an interface.
The class can use various access specifiers like public, private or protected.	The interface makes use of only public access specifier.
The members of a class can be constant or final.	The members of interfaces are always declared as final.

Q.15 Explain how to create an interface in Java with suitable example.

☞ [SPPU : June-22, Marks 3]

Ans. :

Step 1 : Write following code and save it as my_interface.java

```
public interface my_interface
{
        void my_method(int val);
}
```

Do not compile this program. Simply save it.

Step 2 : Write following code in another file and save it using InterfaceDemo.java

Java Program[InterfaceDemo.java]

```
class A implements my_interface {
public void my_method(int i)
{
        System.out.println("\n The value in the class A: "+i);
}
public void another_method() //Defining another method not declared in interface
{
        System.out.println("\nThis is another method in class A");
}
}
class InterfaceDemo
```

Defining the method declared in the interface


```
{
public static void main(String args[])
{
        my_interface obj=new A();
        //or A obj=new A() is also allowed
        A obj1=new A();
        obj.my_method(100);
        obj1.another_method();
        }
}
```

Object of class A is of type my_interface. Using this object method can be accessed

Step 3 : Compile the program created in step 2 and get the following output

```
The value in the class A: 100
This is another method in class A
```

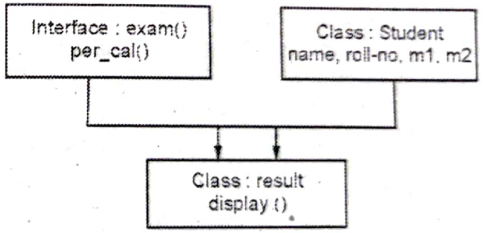
Q.16 Can we achieve multiple inheritance by using interface ? Illustrate with an example.

☞ [SPPU : June-22, Marks 3]

Ans. : Yes we can achieve multiple inheritance by using interface.

Example : Refer Q.17.

Q.17 Write a java program.



Ans. :

```
class Student
{
        String name;
        int roll_no;
        double m1,m2;
        Student(String name,int roll_no,double m1,double m2)
```

```
    {
        this.name=name;
        this.roll_no=roll_no;
        this.m1=m1;
        this.m2=m2;

    }
}
interface exam
{
    public void per_cal();
}
class result extends Student implements exam
{
    result(String nm,int roll,double m1,double m2)
    {
        super(nm,roll,m1,m2);

    }
    public void per_cal()
    {
        double percentage;
        percentage=((m1+m2)/200)*100;
        System.out.println("The percentage marks: "+percentage);
    }
    void display()
    {
        System.out.println("Name: "+name);
        System.out.println("Roll_No: "+roll_no);
        System.out.println("Marks m1= "+m1+" m2 = "+m2);
        per_cal();
    }
}
class Test
{
    public static void main(String args[])
    {
        result r=new result("Parth",10,90,95);
        r.display();
    }
}
```


Output

```
Name: Parth
Roll_No: 10
Marks m1= 90.0 m2 = 95.0
The percentage marks: 92.5
```

4.9 : Polymorphism

Q.18 What is polymorphism ?

Ans. : Definition : Polymorphism is a mechanism in which a single action can be performed in different ways. Here Poly means many and Morphs means forms. Hence polymorphism means many forms.

Q.19 Write a Java program to implement polymorphism

Ans. : Compile Time Polymorphism

- The polymorphism that is resolved during compiler time is known as compile time polymorphism.
- Method overloading is an example of compile time polymorphism.
- The method overloading is a technique in which we use the same function name for different purposes.
- The compile time polymorphism is also called as static polymorphism.
- **Example Program**

```
class calculate {
    int sum(int a,int b) {
        return a+b;
    }
    int sum(int a,int b,int c) {
        return a+b+c;
    }
}
class CompileTimePoly {
    public static void main(String args[]) {
        calculate obj = new calculate();
        System.out.println(obj.sum(10,20));
```

```
        System.out.println(obj.sum(
    }
}
```

Q.20 What is polymorphism ? Illustrate types of polymorphism. [SPPU : June-22, Marks 8]

Ans. : Polymorphism : Refer Q.18.

There are two types of polymorphism :

1. Compile time polymorphism
2. Run time polymorphism

1. Compile time polymorphism : Refer Q.19.

2. Run time polymorphism : The polymorphism which is resolved during run time is called runtime polymorphism.

- Method overriding is an example of run time polymorphism.
- The run time polymorphism is also called as dynamic method dispatch.
- In this technique, the call to method is resolved at run time.
- **Example Program**

```
class Base {
void display() {
System.out.println("\n Base Method Called");
}
}
class Derived extends Base {
void display() //overridden method
{
System.out.println("\n Derived Method Called");
}
}
public class RunPolyDemo {
public static void main(String args[]) {
Base obj=new Derived();//obj is reference to base class
// which is referred by the derived class
obj.display(); //method invocation determined at run time
}
}
```


Output

```
D:\test>javac RunPolyDemo.java
D:\test>java RunPolyDemo
Derived Method Called
```

4.10 : Mechanism for Software Reuse

Q.21 What is software reuse ? Enlist the advantages of software reuse.

Ans. : Definition of software reuse : Software reuse is the process of creating software systems from predefined software components.

Advantages of software reuse

1) Increase software productivity.
2) Shorten software development time.
3) Improve software system interoperability.
4) Develop software with fewer people.
5) Reduce software development and maintenance costs.
6) Produce more standardized software.

4.11 : Efficiency and Polymorphism

Q.22 Write short note on - efficiency and polymorphsim.

Ans. :

- Polymorphism helps programmers reuse the code and classes once written, tested and implemented. They can be reused in many ways.
- Single variable name can be used to store variables of multiple data types
- Polymorphism helps in reducing the coupling between different functionalities.
- One of the disadvantages of polymorphism is that developers find it difficult to implement polymorphism in codes.
- Run time polymorphism can lead to the performance issue as machine needs to decide which method or variable to invoke so it

basically degrades the performances as decisions are taken at run time.

- Polymorphism reduces the readability of the program. One needs to identify the runtime behavior of the program to identify actual execution time.

Q.23 Description : A bank maintains two kinds of accounts for customers. One is saving and other is current. Create a class account that store customer name, account number and type of account. Derived classes with more specific requirements and include necessary methods in order to achieve then following tasks :

i) Accept deposit from the customer and update the balance
ii) Display the balance.
iii) Compute and deposit interest
iv) Permit withdraw and update the balance.
v) Check minimum balance condition and display necessary notice.

☞ [SPPU : June-22, Marks 9]

Ans. : Implementation

```
import java.util.*;
class Account
{
    String cust_name;
    int acc_no;
    double balance;
    String type_of_account;
    Scanner sc;
    public Account()//Destructor
    {
        cust_name="";
        acc_no = 0;
        balance = 0;
        type_of_account ="";
    }
    public void input_data()
    {
        System.out.println("Enter name of Customer");
        cust_name = sc.next();
        System.out.println("Enter account number of Customer");
        acc_no = sc.nextInt();
```



```
        System.out.println("Enter balance: ");
        balance = sc.nextDouble();
        System.out.println("Enter type of account");
        type_of_account = sc.next();
    }
    public void display()
    {
        System.out.println("\n Name: "+cust_name);
        System.out.println("\n Account Number: "+acc_no);
        System.out.println("\nType of Account:
                            "+type_of+account);
        System.out.println("\n Balance Amount:"+balance);
    }
    public void withdraw()
    {
        double amount;
        System.out.println("\n Enter amount to be withdrawn: ");
        amount = sc.nextDouble();
        if( balance >= amount)
        {
            balance = balance - amount;
            System.out.println("Balance: "+balance);
        }
        else
            System.out.println("The amount cannot be drawn");

    }
    public void deposit()
    {
        double amount;
        System.out.println("\n Enter the amount to be deposited: ");
        amount = sc.nextDouble();
        balance = balance + amount;
        System.out.println("Balance: "+balance);
    }
}
class Current extends Account
{
    double servicecharge;
    String chequebook;
```

```
    int penalty;
    public void issueCheque()
    {
        System.out.println("The checkbook issued");
        servicecharge = 20;
        balance = balance - servicecharge;
    }
    public void checkBal()
    {
        System.out.println(balance);
    }
    public void imposePenalty()
    {
        if(balance<=500)
        {
            penalty = 50;
            balance = balance - penalty;
        }
        else
            System.out.println("No Penalty imposed");
    }
}
class Saving extends Account
{
    double interest;
    public void computeInterest()
    {
        int rate;
        rate = 7;
        interest = (amount*no_of_years*rate)/100;
    }
    public void depositInterest()
    {
        balance = balance + interest;
    }
}
class Demo
{
    public static void main(String args[])
    {
```



```
        Saving s = new Saving();
        Current c = new Current();
        s.input_data();
        s.display();
        s.withdraw();

        c.input_data();
        c.display();
        c.deposit();
    }
}
```

END...