Chapter

10 INHERITANCE AND POLYMORPHISM

- *Base And Derived Classes;*
- *Member Access In Derived Classes; Access Specifiers.*
- *Redefinition Of Variables And Functions In Subclasses;*
- *Inheritance and Constructor, Parameterized Constructor.*
- *Inheritance and Overriding.*
- *Abstract Class and Method ; Interface.*

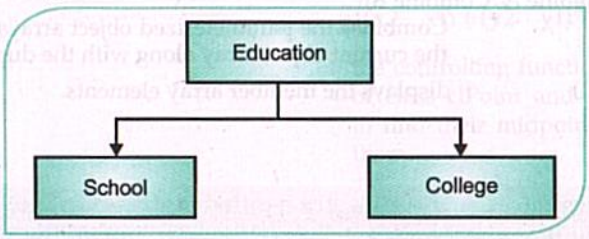
Inheritance

Inheritance is one of the important features of Object Oriented Programming. It allows one class to derive (or share) properties of another class.

The class from which members are taken (the giving class) is called the base (or super or parent) class.

The class which takes members from another class (the taking class) is called derived (or sub or child) class.

For example, we can consider "Education" as the **base** class and "School" and "College" as the **derived** classes. The members of class "Education" are accessible to members of class "School" and class "College".



Advantages of Inheritance

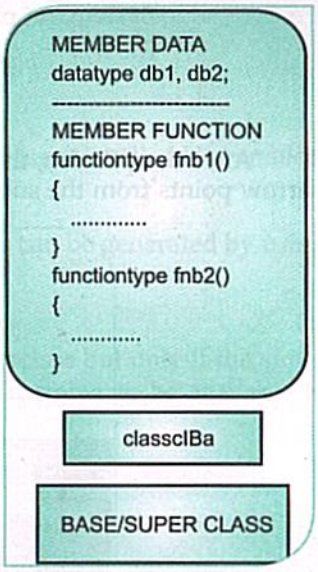
1. **Code Reusability :** The members of one class (base class) can be used by other classes (derived class).

2. **Extensibility :** It allows a task to get extended and cover bigger area of work which can take it upto the highest relevant level of abstraction.

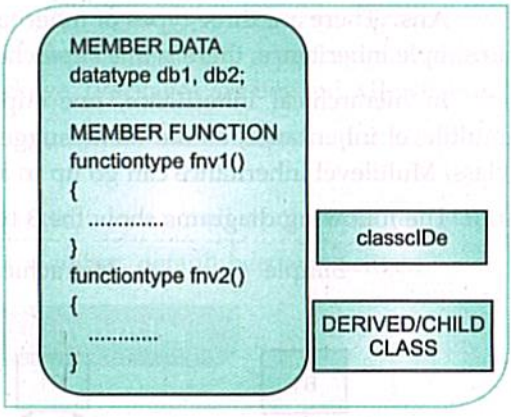
Let us now see programs implementing the concept of inheritance".

Program 1 : A sample program demonstrating the concept and syntax of Inheritance

The base class is clBa and the derived class is clDe.



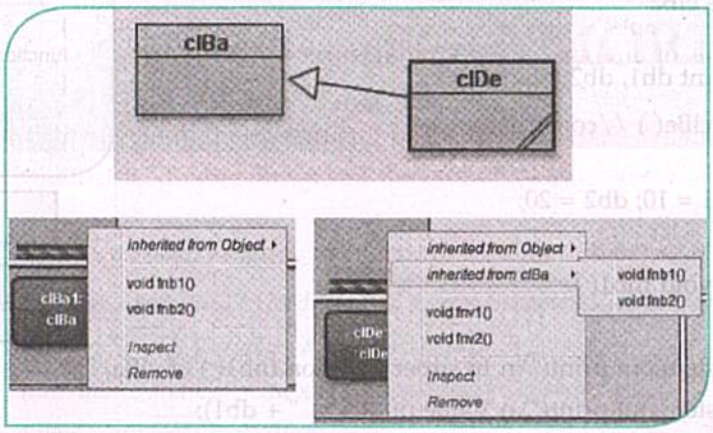

```
import java.io.*;
public class clBa
{
    public int db1, db2;
    public clBa( ) //constructor
    {
        db1 = 10; db2 = 20;
    }
    public void fnb1( )
    {
        System.out.print("\n In super function fnb1( ) of clBa. ");
        System.out.print("\n Value of db1 =  " + db1);
    }
    public void fnb2( )
    {
        System.out.print("\n In super function fnb2( ) of clBa. ");
        System.out.print("\n Value of db2 =  " + db2);
    }
}
```




```
import java.io.*;
public class clDe extends clBa
{
    public int dv1, dv2;
    public clDe( ) //constructor
    {
        dv1 = 500; dv2 = 600;
    }
    public void fnv1( )
    {
        System.out.print("\n In sub function fnv1( ) of clDe. ");
        System.out.print("\n Value of dv1 =  " + dv1 + " and db1 = " + db1 );
    }
    public void fnv2( )
    {
        System.out.print("\n In sub function fnv2( ) of clDe. ");
```

```
        System.out.print("\n Value of dv2 = " + dv2 + " and db2 = " + db2 );
        fnb1();
    }
}
```

After compilation, the project window shows two classes connected with an arrow. The arrow points from the subclass to the superclass.



Two objects have been created, one for the superclass clBa named clBa1 and the other for the sub class clDe named clDe1.

The object of the sub class gets access to its own members (data and methods) and also to the members of its super class.

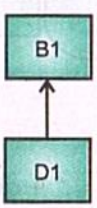
Q. 1. What are the types of inheritance in Java?

Ans. There are three types of inheritance in Java – Simple, Hierarchical and Multilevel. In simple inheritance, there is one superclass from which one or more subclasses are derived.

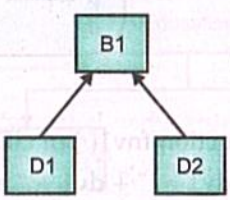
In hierarchical inheritence, one super class has more than one direct subclass. In multilevel inheritance, as the name suggests, a subclass becomes superclass of some other class. Multilevel inheritance can go up to as many number of levels.

The following diagrams show the 3 types of inheritance :

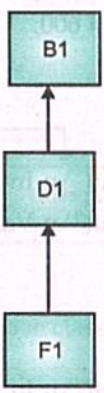
Simple



Hierachical



Multi Level



Note :

- *There is another kind of inheritance called multiple inheritance, in which a sub class inherits properties of more than one super class.*

When we inherit more than one derive class from single base class, it is called Hierarchical inheritence.

Java does not support multiple inheritance, but similar effects can be generated by using interface. [Details of interface is shown in next section.]

Q. 2. What are access specifiers?

Ans. Members of a sub-class can access members of its superclass but only if the super class keeps those members available. Access specifiers permit a member to be available to other classes and to what extent.

Access Control of Class Members

Access Specifiers	In the same class	In the same Package		In another Package		Objects
		In Derived Class	In non Derived Class	In Derived Class	In non Derived Class	
private	Y	N	N	N	N	N
protected	Y	Y	Y	Y	N	Y
public	Y	Y	Y	Y	Y	Y
default	Y	Y	Y	N	N	N

1. A package is a collection of classes. The classes belonging to a package may have a super-class in the same package or in another package. [Packages are beyond XII syllabus.]
2. Objects of a class can access only those members which are present under public access modifier.
3. Java allows only single/multilevel inheritance. It does not allow multiple inheritance.
4. When no access specifier is specified for a member, default becomes active.

Inheritance and Constructor

Constructor

Constructor is a block of code, which runs when keyword new is used to **instantiate an object.** It looks like a method, however it is not exactly a method. Methods have return type but constructors don't have any return type. It is used to initialise the data members with initial values.

How to call a constructor?

The constructor gets called when we create an object of a class (i.e., new keyword followed by class name). Constructors can get executed only once. For e.g., Demo obj= new Demo(); (here Demo() is a default constructor of Demo class).

Default constructor : It is also known as no-argument constructor. Constructor with no arguments is known as default constructor. It is automatically provided by java compiler when it is not present explicity

```
class Demo
{
    public Demo()
    {
        System.out.println("This is a default constructor");
    }
}
```

Parameterized constructor : Constructor with argument list is known as parameterized constructor class Demo

```
{
    public Demo(int num, String str)
    {
        System.out.println("This is a parameterized constructor");
    }
}
```

In inheritance, a sub class cannot inherit the Constructor of its super class. But the Constructor of the super class gets executed automatically and that too before the Constructor of the sub class whenever an object of the sub class is created.

The following example explains the above statement.

Program 3 : In Program1, we modify the constructors of both the Base and Derived class and add a print statement in each.

```
    public clBa( ) //constructor
    {
        System.out.print("\n constructor of Base Class clBa ");
        db1 = 10; db2 = 20;
    }
    public clDe( ) //constructor
    {
        System.out.print("\n constructor of Derived Class clDe. ");
        dv1 = 500; dv2 = 600;
    }
```

If we now create an object of the base class clBa, the output will be

constructor of Base Class clBa.

And if we create an object of the derived class clDe, the output will be

constructor of Base Class clBa.

constructor of Derived Class clDe.

Inheritance and Paramterized Constructor

In case of parameterized constructors, the parameters of the Constructor of the super lass have to be assigned from the sub class. The reason behind this is that the super class onstructor gets executed before the sub class Constructor, when an object of the sub class created.

To send arguments to the super class Constructor, keyword **super** is used.The following rogram shows the syntax.

Program 4 :

```
import java.io.*;
public class clBase
{
    public int db1, db2;
    public clBase (int p1, int p2 ) //constructor
    {
        System.out.print("\n Parameterized  Constructor of Base Class clBase :  ");
        db1 = p1; db2 = p2;
    }
    public void fnB( )
    {
        System.out.print("\n Class clBase ::  Method fnB ( ) :  ");
        System.out.print("\n Value of db1 =  " + db1 +   " Value of db2 = " + db2);
    }
}
import java.io.*;
public class clDerive extends clBase
{
    public int dv1, dv2;
```

```
    public clDerive (int q1, int q2, int q3, int q4 ) //constructor
    {
        super(q1, q2); //sending values to the super class
        System.out.print("\n Parameterized  Constructor of Derived Class clDe :  ");
        dv1 = q3 ;  dv2 = q4 ;
    }
    public void fnD( )
    {
        System.out.print("\n Class clDerive ::  Method fnD ( ) :  ");
        System.out.print("\n Base Class Data db1 =  " + db1 +   " Value of db2 = " + db2);
        System.out.print("\n Derived Class Data dv1 =  " + dv1 +   " Value of dv2 = " + dv2);
    }
}
```

The Parameterized Constructor of the sub class executes its own statements only after it sends arguments to its super class Parameterized Constructor. Therefore, sending arguments through **super** must be the first statement in the sub class Parameterized Constructor.

Shown below is the object creation of the derived class clDe, its constructor accepting parameters and sequence of output statements.

Application of Parameterized Constructors in both super and sub class

Program 5 : A library issues books and charges 2% of the cost price (cp) per day. After 7 days, if the book is not returned, fine is charged for extra days as per the following:

Number of excess days	Fine per day (Rs)
1 – 5	2
6 – 10	3
Above 10	5

Design a class **Library,** the details of which are given below :

Class Name	Library
Member Data	
name	String
author	String
cp	Int
Member Function	
Library(...)	Parameterized Constructor to initiate member data

void show ()	to display book details

Design another class **Compute,** that is inherited from class Library, the details of whose are given below :

Class Name	Compute
Member Data	
d	int, to store total number of days
f	int, to store fine if applicable
Member Function	
Compute(...)	Parameterized Constructor to initiate member data of both this class and its base class (3 base class, 1 this class)
void fine ()	to calculate fine
void display ()	to display the amount payable

```
import java.io.*;
public class Library
{
    String name, author; intcp;
    public Library(String cname, String cauthor, int ccp)
    {
        name = cname;   author = cauthor; cp = ccp;
    }
    public void show()
    {
        System.out.print("\n Name : " + name + "  Author : " + author + "  Cost price
                                                                = Rs." + cp);
    }
}
import java.io.*;
public class Compute extends Library
{
    int d, f;
    public Compute(String cname, String cauthor, intccp, int cd)
    {
        super(cname, cauthor, ccp);  // to send values to parameterized constructor of the
                                                                //base class Library
        d = cd;
        f = 0;
    }
```

```
    public void fine()
    {
        if(d >=1 && d<=5)
            f = 2*d;
        else if(d<=10)
            f = 2*(5) + 3*(d–5);
        else
            f = 2*(5) + 3*(5) + 5*(d–10);
    }
    public void display()
    {
        System.out.print("\n Amount payable is " + f);
    }
}
```

Inheritance and Overriding

Q. 1. What is overriding?

Ans. Declaring a method in subclass which is already present in parent class with same signature is known as method overriding. It is a feature that allows both super and sub class to contain members (data/method) having the same name. This feature gives the sub class a scope to hide members of its base class.

In case of overridden members, the object of the sub class can instantiate only those that belong to the sub class itself.

In case the base and the derived class both contain certain members (data/method) that share same names, the objects of the derived class can instantiate only those belonging to the derived class.

The objects of the sub class can instantiate those members of the super class by using keyword **super.**

The syntax is **super.baseClassMethodName( ).**

Example :

One of the simplest example – Here Boy class extends Human class. Both the classes have a common method void eat(). Boy class is giving its own implementation to the eat() method or in other words it is overriding the method eat().

```
class Human
{
    public void eat()
    {
        System.out.println("Human is eating");
    }
}
class Boy extends Human
```

```
{
    public void eat()
    {
        System.out.println("Boy is eating");
        {
            public static void main( Stringargs[])
            {
                Boyobj = newBoy();
                obj.eat();
            }
        }
    }
}
```

Output :

Boyis eating

Program 6 : In a bank, the account holders were offered two kinds of interest scheme, Simple and Compound, both depending on Principal (P), Rate (R) and Time (T). The formula for calculating Simple Interest is given as SI = (PxRxT)/100. And to calculate Compound Interest, formula used is $CI = Px(1 + R/100)^T - P$.

The bank manager wishes to implement a program whose structure is given below. A class **ACCOUNT** has two sub classes SIMPLE and COMPOUND, as given below :

Class Name	ACCOUNT
Member Data	
P	double, to store the Principal
R	double, to store the Rate
T	int, to store the time
Member Function	
Account (...)	Parameterized Constructor to initiate member data
void display ()	to display the member data

Class **SIMPLE,** that is inherited from class **ACCOUNT**, the details of whose are given below :

Class Name	SIMPLE
Member Data	
Si	double, to store the value of simple interest
Member Function	
Simple (...)	Parameterized Constructor to initiate member data of both this class and its base class (3 base class, member of this class to get initialized to 0)
void Interest ()	to calculate simple interest
void display ()	to display the amount payable and also the values of P, R and T.

Class **Compound,** that is inherited from class Account, the details of whose are given below :

Class Name	COMPOUND
Member Data	
Ci	double, to store the value of compound interest
Member Function	
Compound (...)	Parameterized Constructor to initiate member data of both this class and its base class (3 base class, member of this class to get initialized to 0)
void Interest ()	to calculate compound interest
void display ()	to display the amount payable and also the values of P, R and T.

In this example, display( ) is an overridden method.

The program is shown below :

```
class ACCOUNT
{
    protected double P, R, T ;
    public ACCOUNT(double aP, double aR, double aT)
    {
        P = aP; R = aR; T = aT;
        System.out.print("\n\n Constructor of ACCOUNT.");
    }
    public void display ()
    {
        System.out.print("\n\n From display() of class ACCOUNT ..\n  P = " + P +
                                                    " R = " + R  + " T = " + T);
    }
}
class SIMPLE extends ACCOUNT
{
    double si;
    public SIMPLE (double sP, double sR, double sT)
    {
        super(sP, sR, sT);
        si = 0;
        System.out.print("\n\n From constructor of SIMPLE  si = " + si);
    }
```

```
    public void Interest()
    {
        si = (P * R * T)/100;
    }
    public void display ()
    {
        super.display(); //otherwise the present method will get a call resulting in
                                                    infinite recursion Interest();
        System.out.print("\n\n From display() of SIMPLE  si = " + si);
    }
}
class COMPOUND extends ACCOUNT
{
    double ci;
    public COMPOUND (double cP, double cR, double cT)
    {
        super(cP, cR, cT);
        ci = 0;
        System.out.print("\n\n From constructor of COMPOUND  ci = " + ci);
    }
    public void Interest()
    {
        ci = (P * Math.pow((1 + R/100), T) )–P;
    }
    public void display ()
    {
        super.display(); //otherwise the present method will get a call resulting in
                                                    infinite recursion Interest();
        System.out.print("\n\n From display() of COMPOUND  ci = " + ci);
    }
}
```

OUTPUT :

```
P = 1000
R = 5
T = 4
// obj creation of SIMPLE
// fn call of display() from SIMPLE
```

```
Constructor of ACCOUNT.
From constructor of SIMPLE  si = 0.0
From display() of the ACCOUNT ..
P = 1000.0 R = 5.0 T = 4.0
From display() of SIMPLE  si = 200.0
// obj creation of COMPOUND
// fn call of display() from COMPOUND
Constructor of ACCOUNT.
From constructor of COMPOUND  ci = 0.0
From display() of the ACCOUNT ..
P = 1000.0 R = 5.0 T = 4.0
From display() of COMPOUND  ci = 215.50625000000014
```

Abstract Class & Abstract Method

What is an ABSTRACT CLASS ?

An abstract class is a special kind of class which is used in huge programs for implementing uniformity and standard. An abstract class has its members (data and methods) but it cannot be instantiated (it cannot have objects). Other classes can be inherited from an abstract class and its sub classes can be instantiated. Objects of these sub classes can invoke methods of its own and the abstract super class.

What is an ABSTRACT METHOD?

An abstract method is a special kind of method that contains no statement. Abstract method of a class has to be defined in its subclass where the method statements are to be provided.

Abstract methods can be contained by abstract classes only.

To declare an abstract class and method, keyword abstract is used (shown in the next example).

Note :

- *Need of abstract class.*

In a team of members working on a project, the team leader has to plan the various parts. The team leader can create abstract classes for his team members to implement. The methods which the team leader decides must be present in all the subclasses, he can declare them to be abstract methods.

In this way, the team leader can divide the project into his team members where he can insist certain structure and uniformity is maintained by all his team members for sure.

The presence of an abstract method is like a necessary reminder to its derived classes. The abstract method will have to be present in all the derived classes with the relevant statements.

Observe the following example :

Program 7 : A Class SHAPE is abstract. It has two derived classes, CIRCLE and RECTANGLE. The detail of the classes is given below :

Abstract Class Name	clShape
Member Data	
name	String to store the name of the shape
Member Function	
clShape (String sname)	Parameterized Constructor to initiate member data
void Display ()	to display the member data
void Area()	Abstract method for area calculation

Class **clCircle,** that is inherited from class **clShape,** the details of whose are given below :

Class Name	clCircle
Member Data	
r	int, to store the radius
Area	double, to store the area
Member Function	
clCircle (String cname, intcr)	Parameterized Constructor to initiate member data
void Display ()	to display the member data of both classes
void Area()	For area calculation

Class **clRectangle,** that is inherited from class **clShape,** the details of whose are given below :

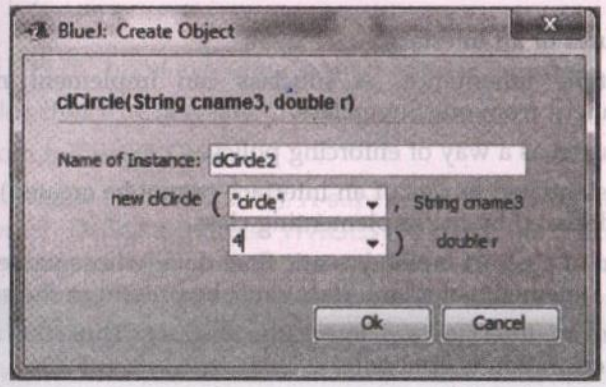
Class Name	clRectangle
Member Data	
l, b	double , to store the length and breadth
Area	double, to store the area
Member Function	
clRectangle (String cname, double cl, double cb)	Parameterized Constructor to initiate member data
void Display ()	to display the member data of both classes
void Area()	For area calculation

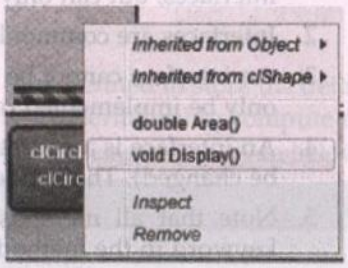
```
public abstract class clShape
{
    protected String name;
    clShape(String cname)
    {
        name = cname;
    }
    public  void Display()
    {
        System.out.print(" Name of the Shape is : " + name);
    }
    public abstract double Area(); //abstract class
}
classclRectangle extends clShape
{
    protected double length, breadth;
    clRectangle(String cname2, double cl, double cb)
    {
        super(cname2);
        length = cl; breadth = cb;
    }
    public double Area()
    {
        return (length * breadth) ;
    }
    public void Display ()
    {
        super.Display();
        System.out.print("\nThe length is : " + length + "  breadth is : " + breadth + " &
                                                        area is : " + Area());
    }
}
classclCircle extends clShape
{
    protected double radius;
    clCircle(String cname3, double r)
    {
        super (cname3);
```

```
        radius = r;
    }
    public double Area()
    {
        return 3.14 * radius * radius ;
    }
    public void Display()
    {
        super.Display();
        System.out.print("\nThe radius is : " + radius + " and area is : " + Area());
    }
}
```

Execution :

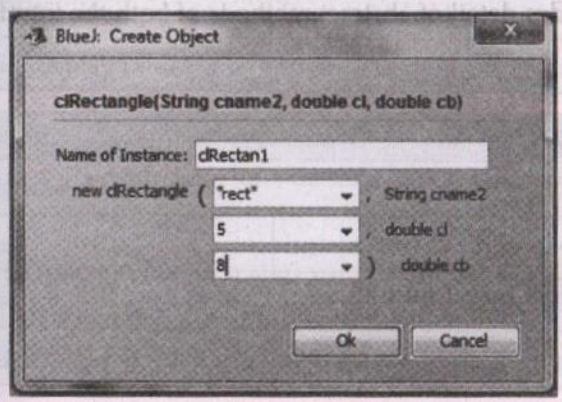
1. Creating an object of class clCircle. Shown below is the constructor executing step, and then executing method Display().

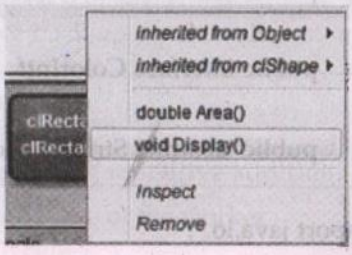




Name of the Shape is : circle
The radius is : 4.0 and area is : 50.24

2. Creating an object of class clCircle. Shown below is the constructor executing step, and then executing method Display().





Name of the Shape is :rect

The length is : 5.0 breadth is : 8.0 & area is : 40.0

Note :

1. *It is not possible to create an object of type* ***clShape.***
2. *All subclasses of* ***clShape*** *must override abstract method* ***Area( )****. To prove this to yourself, try creating a subclass that does not override method* ***Area( )****. You will receive a compile-time error.*
3. *Display() is also an overridden method.*
4. *If a class includes abstract methods, the class itself must be declared* ***abstract.***

Interface

What is an Interface?

An interface is a kind of class that can contain only abstract methods and static final data. As mentioned above, Java does not support multiple inheritance. Interfaces give a way to apply the benefits of multiple inheritance. A class can derive the properties of more than one interface.

Given below are certain features of an Interface.

1. Interfaces simulate multiple inheritance. A subclass can implement multiple interfaces, but can only inherit from one Superclass.
2. Interfaces are commonly used as a way of enforcing policy.
3. An interface cannot be instantiated (object of an interface cannot be created) . It can only be implemented or declared by the implementing class.
4. An interface is a collection of abstract methods static final data (whose value cannot be changed). The method statements of an interface must be present in the subclass.
5. Note that all methods in an Interface are implicitly abstract. Thus the abstract keyword in the method declaration is optional.
6. A class may implement more than one interface. In that case the sub class has to provide details of all the methods of all the interfaces. Interface names are to be separated by comma.

The following program shows the implementation of two interfaces in a sub class.

Program 8 : A program having two interfaces DesignIntf and ColorIntf. A class Furniture implements both the interfaces, provides detail of abstract methods of both the interfaces.

```
public interface DesignIntf
{
    public abstract void fnDesignChoice();
}
    public interface ColorIntf
{
    public  abstract String fnColorChoice( int n);
}
import java.io.*;
```

```
public class Furniture implements ColorIntf, DesignIntf
{
    //to givestmts to the methods of its parent interfaces
    public String fnColorChoice(int k)
    {
        if (k==1)
            return "Brown";
        return "Chocolate";
    }
    public void fnDesignChoice()
    {
        System.out.print("\n From fn Design Choice ");
    }
    public void main()
    {
        System.out.print(" color choice : " + fnColorChoice(2));
        fnDesignChoice();
    }
}
```

OUTPUT : upon executing main () of class Furniture.

color choice : Chocolate

From fn Design Choice

Program 9 : Qn [ISC 2010]. A super class WORKER has been defined to store the details of a worker. Define a subclass WAGES (by using the concept of Inheritance) to compute the monthly wages for the worker. The details/specifications of both the classes are given below :

Class Name	WORKER
Member Data	
Name	String - to store the name of the worker
Basic	Double - to store the basic pay in decimals
Member Functions	
WORKER(...)	Parameterized Constructor to initiate instance variables
void display ()	to display the workers details

Class Name	WAGES
Member Data	
hrs	int - to store the hours worked
rate	double - to store the rate per hour
wage	double - to store the overall wage of the worker

Member Functions	
WAGES(...)	Parameterized Constructor to initiate instance variables of both classes
double overtime ()	Calculates and returns the overtime amount as (hours * rate)
void display()	Calculates the wage using the formula wage = overtime amount + Basic pay and displays it along with the other details

Given below is the detail of the constructor of the derived class –

```
WAGES ( Stringdname, double dbasic, int dhrs, double drate)
{
    super ( dname, dbasic); // parameterized constructor of the base class is sent values
    hrs = dhrs;
    rate = drate ;
    wage = 0;
}
```

Program 10 : A Travel Agency manager decided to implement some auto upgrading applications. For that purpose, the following classes were created. Given below are some parts of the entire project.

Class **TRAVEL** and class **JOURNEY** were created such that **JOURNEY** was a **derived (or inherited)classof TRAVEL.**

Details of class **TRAVEL** is given below :

Class name	TRAVEL
Data members/instance variables	
pName – to store name of a passenger	Data-type is String
pAge – to store age of a passenger	Data-type is int
pFone – to store the phone number	Data-type is long
Member functions/methods :	
TRAVEL(...)	Parameterized Constructor to fill member data.
void fnInput (void)	To fill member data.
void fnOutput (void)	To show member data.

Details of derived class **JOURNEY** is given below :

Class name	JOURNEY
Data members/instance variables	
fromCity – to store starting city name	Data-type is String

toCity – to store final city name	Data-type is String
fare – to store cost of travel	Data-type is double
Member functions/methods :	
JOURNEY (...)	Parameterized Constructor to fill member data of both derived and base class
void fnInput (void)	To fill member data.
void fnOutput (void)	To show all data. Also print the amount that a passenger should pay considering if he is eligible for any discount or not. [Hint: This function may call another function.]
double fnDiscount(void)	To calculate and return a discount of 40% on cost of travel (fare) if the age of the passenger is <12 years and a discount of 60% if the age>=60 years.

Specify the classes **TRAVEL** and its derived class **JOURNEY** by giving the above mentioned details.

Given below is the detail of the parameterized constructor of the derived class :

```
JOURNEY ( Stringj Name, int jAge, long jFone, String jFromCity, String jToCity, double
        jFare )
{
    super ( jName,  jAge, jFone);
    fromCity = jFromCity;
    toCity = jToCity;
    fare = jFare;
}
```

[This program is done above (program number 5) that implemented parameterized constructors for both classes. Here the program shows with function overriding]

Program 11 : A library issues books on rental basis at a 2% charge on the cost price of the book per day. A book can be retained for 7 days without fine. If the book is returned after 7 days, a fine will also be charged for the excess days as per the chart :

Excess Days	Fine (Rs)
1 – 5	2.00
6 – 10	3.00
Above 10	5.00

A class **Library** and **Compute** has been designed [**library**-base class and **compute**–derived class]

Class name	LIBRARY
Data members/instance variables	
name	Name of book
author	Name of author
p	Price of the book in decimal fractions
Member functions/methods :	
LIBRARY(...)	Parameterized Constructor to fill member data.
void display (void)	To display book details (all)

Class name	COMPUTE
Data members/instance variables	
d	Number of days taken in returning the book
f	to store the fine
Member functions/methods :	
COMPUTE (...)	Parameterized Constructor to fill member data of both derived and base class
void fine ()	To calculate the fine for the extra days
void display ()	To display book details , Fine(if any), total amount payable [hint – 2% of book price * num of days + fine]

Short Questions

Q. 1. Does Java support Multiple Inheritance ? Justify.

Ans. No, Java doesn't support multiple inheritance. Interfaces doesn't facilitate inheritance and hence implementation of multiple interfaces doesn't make multiple inheritance.

Q. 2. Why java doesn't support multiple Inheritance ?

Ans. Multiple Inheritance can cause certain ambiguities in a program.

```
class A
{
    void test()
    {
        System.out.println("test() method");
    }
}
```

```
class B
{
    void test()
    {
        System.out.println("test() method");
    }
}
class C extends A, B // Assuming Java permitted multiple inheritance
{
        //code
}
```

Here metodtest() is present in both classess A, B which may have different code for themselves. The ambiguity in this form of multiple inheritance is for class C to have the method test().

Java does not support multiple inheritance because it causes ambiguity.

Q. 3. Are constructors inherited? Can a subclass call the parent's class constructor? When?

Ans. You cannot inherit a constructor. That is, you cannot create a instance of a subclass using a constructor of one of its superclasses. One of the main reasons is because you probably don't want to override the superclasses constructor, which would be possible if they were inherited. By giving the developer the ability to override a superclasses constructor you would erode the encapsulation abilities of the language.

Q. 4. What is the difference between keyword *this* and *super()* ?

Ans. Keyword **this** is a reference to the current object in which this keyword is used whereas super is a reference used to access members specific to the parent Class. Keyword this is primarily used for accessing member variables if local variables have same name, for constructor chaining and for passing itself to some method whereas **super** is primarily used to initialize base class members within derived class constructor.

Q. 4. What will happen if class implements two interface having common method?

Ans. Methods of interface contain only name whose code has to be filled by the implementing class. If a class implements two interfaces, where both have a common method, the detail of that method has to be filled in this class. Hence there would be no ambiguity.

Q. 5. What are points to consider in terms of access modifier when we are overriding any method?

Ans. 1. Overriding method cannot be more restrictive than the overridden method.

Reason : in case of polymorphism , at object creation JVM look for actual runtime object. JVM does not look for reference type and while calling methods it look for overridden method.

If by means subclass were allowed to change the access modifier on the overriding method, then suddenly at runtime—when the JVM invokes the true object's version of the method rather than the reference type's version then it will be problematic.

2. In case of subclass and superclass define in different package, we can override only those method which have public or protected access.

3. We can not override any private method because private methods can not be inherited and if method can not be inherited then method can not be overridden.

Q. 6. What is covariant return type?

Ans. Co-variant return type states that return type of overriding method can be subtype of the return type declared in method of superclass. It has been introduced since jdk 1.5.

Q. 7. How compiler handles the exceptions in overriding ?

Ans. 1. The overriding methods can throw any runtime Exception , here in the case of runtime exception overriding method (subclass method) should not worry about exception being thrown by superclass method.

2. If superclass method does not throw any exception then while overriding, the subclass method can not throw any new checked exception but it can throw any runtime exception.

3. Different exceptions in java follow some hierarchy tree(inheritance). In this case if superclass method throws any checked exception , then while overriding the method in subclass we can not throw any new checked exception or any checked exception which are higher in hierarchy than the exception thrown in superclass method.

Summary of Inheritance

1. A class can have exactly one direct Superclass.

2. A class can hide data and methods of its Superclass, by overriding them. In this case the members of the superclass can be accessed by using the keyword super.

3. You can **prevent a class from being subclassed** by using the **final** keyword in the class's declaration. Similarly, you can **prevent a method from being overridden** by subclasses by declaring it as a **final method.**

4. An abstract class can only be subclassed; it cannot be instantiated. An abstract class can contain abstract methods—methods that are declared with an empty body (no statements). Subclasses then provide the implementations for the abstract methods.

Exercise

1. State TRUE or FALSE. Correct the false statements.

(a) The class from which members are taken is called the base class.

(b) Access Specifier private allows access to members of the same class.

(c) Access Specifier default allows the objects to access to members of the class.

(d) Constructor of the derived class gets executed before the constructor of the base class.

(e) Overriding does not allow base and derived class to have members of same name.

2. Answer the following in brief, with a program code example.

(a) What is overriding ?

(b) In inheritance, what is the order in which the constructors are executed ?

(c) What are the two applications of keyword *super* in inheritance ?

(d) Does objects get access to member for all access sepcifiers ?

(e) What is an abstract method ?

Answer the questions given below the following codes :

A.

```
class SIMPLE extends ACCOUNT
{
    doublesi;
    public SIMPLE (double sP, double sR, double sT)
    {
        super(sP, sR, sT);
        si = 0;
        System.out.print("\n\n From constructor of SIMPLE  si = " + si);
    }
    public void Interest()
    {
        si = (P * R * T)/100;
    }
    public void display ()
    {
        super.display();
        Interest();
        System.out.print("\n\n From display() of SIMPLE  si = " + si);
    }
}
```

(a) Name the base class and the derived class.

(b) Write the prototype of the constructor of the base class.

(c) Write the keyword that is connecting the two classes.

(d) Name the variables of the base class that has been used here.

(e) What is the use of *super.display()* ? What will happen without keyword *super* here?

B.

```
class JOURNEY extends PASSENGER
{
    String cityFrom, cityTo;
    double fare , discount ;
    public JOURNEY (String name2, int age2, String cityFrom1, String cityTo1, double fare1)
    {
        super ( name2, age2) ;
```

```
        cityFrom = cityFrom1 ;
        cityTo = cityTo1 ;
        fare = fare1 ;
        discount = 0;
    }
}
public void fnDiscount ( )
{
    if ( super.age>= 60  ||  (super.age<=12  &&super.age>=5 ) )
    {
        discount = 0.5 * fare ;
    }
    else if ( super.age<= 5)
    {
        discount = fare ;
    }
}
public void display ()
{
    super.show ( ) ;
    fnDiscount() ;
    System.out.println( name + " is travelling from " + cityFrom + " to " + cityTo ) ;
    System.out.println( " Age of passenger is " + age ) ;
    System.out.println( " Ticket amount payable is " + ( fare – discount) ) ;
}
```

(a) Name the base class and the derived class.

(b) Write the prototype of the constructor of the base class.

(c) Name the variables of the base class that has been used here.

(d) Consider no other member is present in the derived class, what will happen without keyword *super* in the method call *super.show() ?*

(e) Consider *age* is the member data of the base class. Does it have to accessed using *super.age* in the derived class methods ?

4. A super class WORKER has been defined to store the details of a worker. Define a sub class WAGES to compute the monthly wages for the worker. The details of both the classes are given below :

Class name : WORKER

Data members :

Name : To store the name of the worker.

Basic : To store the basic pay in decimal.

Member methods :

Worker(....)	:	Parameterized constructor to assign values to the instance variables.
void display()	:	Display worker details
Class name	:	WAGES
Data members :		
hrs	:	To store the hours worked
rate	:	To store rate per hour
wage	:	To store the overall wage of the worker
Member methods :		
Wages(....)	:	Parameterized constructor to assign values to the instance variables of both classes.
double overtime( )	:	Calculates and returns the overtime amount as (hours * rate).
void display()	:	Calculates the wage and displays it along with other details. Formula to calculate wage is given as wage = overtime amount + basic pay

Specify the class WORKER giving details of the constructor() and void display().

Using the concept of Inheritance, make class WAGES a derived class of WORKER. Write detail of the specified methods only.

A super class DETAIL has been defined to store the details of a customer. Define a sub class BILL to compute the monthly telephone charge of the customer as per the chart given below :

Number of Calls	Rate
1- 100	Only rental charge
101-200	60 paisa per call + rental charge
201-300	80 paisa per call + rental charge
Above 300	1 rupee per call + rental charge

The details of both the classes are given below :

Class Name : DETAIL

Data members :

name	:	To store the name of the customer.
address	:	To store the address of the customer.
telno	:	To store the phone number of the customer.
rent	:	To store the monthly rental charge.
Member functions :		
Detail(..)	:	Parameterized constructor to assign values to data members.
void show()	:	To display the detail of the customer.
Class Name	:	BILL
Data members :		
n	:	An integer to store the number of calls.
amt	:	A double type data to store the amount to be paid by the customer.

Member functions :

Bill(..)	:	Parameterized constructor to assign values to data Members of both classes and to initialize amt = 0.0.
void calculate ()	:	Calculates the monthly telephone charge as per the rate chart given above

void show() : To display the detail of the customer and amount to be paid

Specify the class DETAIL and make class BILL its derived class.

Write the details of the methods mentioned in the question.

6. A super class RECORD has been defined to store the names and ranks of 50 students. Define a sub class RANK to find the highest rank along with the name. The details of both classes are given below :

Class name	:	RECORD
Data members :		
name[]	:	To store the names of students
rnk[]	:	To store the ranks of students
Member functions :		
RECORD()	:	Constructor to initialize data members
voidreadValues()	:	To store names and ranks
void display()	:	Displays the names and the corresponding ranks
class name	:	RANK
Data members	:	index : An Integer to store the index of the topmost rank
Member functions :		
RANK()	:	Constructor to invoke the base class constructor and to initialize index to 0.
void highest()	:	Finds the index location of the topmost rank and stores it in index without sorting the array
void display()	:	Displays the name and ranks along with the name having the topmost rank.

Specify the class RECORD and make RANK its subclass.

Write detail of the mentioned methods.