**JAVA**

**Inheritance**

**Lab Exercise No:**34

**Exercise Objective(s):***The concept of inheritance*

**Exercise:***Create a class called Vehicle. Create subclasses like Truck, Bus, Car etc. Add common methods in the base class and specific methods in the corresponding class. Create a class called Road*

*and create objects for the Truck, Car, Bus etc and display the appropriate message.*

**Solution:**

**package** hsbc.training.day3;

**public** **class** Roads

{

**public** **static** **void** main(String[] args)

{

//creating Truck object

Truck t=**new** Truck();

t.TruckDetails();

//creating Bus object

Bus b=**new** Bus();

b.BusDetails();

//creating Car object

Car c=**new** Car();

c.CarDetails();

//creating Bike object

Bike bk=**new** Bike();

bk.BikeDetails();

}

}

//class Vehicle is the base/super class

**class** Vehicle

{

//display vehicle details

**protected** **void** vehicleDetails(**int** regNo,String brand)

{

System.***out***.println("Registration Number ==>"+regNo);

System.***out***.println("Brand Name ==>"+brand);

}

}

//subclass of Vehilce

**class** Truck **extends** Vehicle

{

//display truck details by calling superclass method

**public** **void** TruckDetails()

{

System.***out***.println("-------Truck Details--------");

vehicleDetails(12563,"Mahindra");

}

}

//subclass of Vehilce

**class** Bus **extends** Vehicle

{

//display bus details by calling superclass method

**public** **void** BusDetails()

{

System.***out***.println("-------Bus Details--------");

vehicleDetails(88563,"Tata Motors");

}

}

//subclass of Vehilce

**class** Car **extends** Vehicle

{

//display car details by calling superclass method

**public** **void** CarDetails()

{

System.***out***.println("-------Car Details--------");

vehicleDetails(42063,"Audi");

}

}

//subclass of Vehilce

**class** Bike **extends** Vehicle

{

//display bike details by calling superclass method

**public** **void** BikeDetails()

{

System.***out***.println("-------Bike Details--------");

vehicleDetails(12500,"Honda");

}

}

**Lab Exercise No:**35

**Exercise Objective(s):***super keyword*

**Exercise:***In the Lab Exercise 34, in the Vehicle class constructor initialize few variables like color, no of*

*wheels, model etc. Give appropriate values for these variables from the invoking subclass.*

**Solution:**

**package** hsbc.training.day3;

**public** **class** NewRoads

{

**public** **static** **void** main(String[] args)

{

//creating Truck object

Truck t=**new** Truck(12345,"Mahindra",6,"Orange");

t.TruckDetails();

//creating Bus object

Bus b=**new** Bus(54321,"Tata Motors",4,"Red");

b.BusDetails();

//creating Car object

Car c=**new** Car(9876,"Audi",4,"Black");

c.CarDetails();

//creating Bike object

Bike bk=**new** Bike(56789,"Honda",2,"Red");

bk.BikeDetails();

}

}

//class Vehicle is the base/super class

**class** Vehicle

{

**protected** String color,brand;

**protected** **int** regNo,noOfWheels;

//Parameterized constructor of Vehicle class

Vehicle(**int** regNo,String brand,**int** noOfWheels,String color)

{

**this**.regNo=regNo;

**this**.brand=brand;

**this**.noOfWheels=noOfWheels;

**this**.color=color;

}

//display vehicle details

**protected** **void** displayDetails()

{

System.***out***.println("Registration Number ==>"+regNo);

System.***out***.println("Brand Name ==>"+brand);

System.***out***.println("Color ==>"+color);

System.***out***.println("No. of Wheels ==>"+noOfWheels);

}

}

//subclass of Vehicle

**class** Truck **extends** Vehicle

{

//Truck constructor calling Vehicle Constructor

Truck(**int** regNo, String brand, **int** noOfWheels, String color)

{

**super**(regNo, brand, noOfWheels, color);

}

//display truck details by calling superclass method

**public** **void** TruckDetails()

{

System.***out***.println("-------Truck Details--------");

displayDetails();

}

}

//subclass of Vehilce

**class** Bus **extends** Vehicle

{

//Bus constructor calling Vehicle Constructor

Bus(**int** regNo, String brand, **int** noOfWheels, String color)

{

**super**(regNo, brand, noOfWheels, color);

}

//display bus details by calling superclass method

**public** **void** BusDetails()

{

System.***out***.println("-------Bus Details--------");

displayDetails();

}

}

//subclass of Vehilce

**class** Car **extends** Vehicle

{

//Car constructor calling Vehicle Constructor

Car(**int** regNo, String brand, **int** noOfWheels, String color)

{

**super**(regNo, brand, noOfWheels, color);

}

//display car details by calling superclass method

**public** **void** CarDetails()

{

System.***out***.println("-------Car Details--------");

displayDetails();

}

}

//subclass of Vehilce

**class** Bike **extends** Vehicle

{

//Bike constructor calling Vehicle Constructor

Bike(**int** regNo, String brand, **int** noOfWheels, String color)

{

**super**(regNo, brand, noOfWheels, color);

}

//display bike details by calling superclass method

**public** **void** BikeDetails()

{

System.***out***.println("-------Bike Details--------");

displayDetails();

}

}

**Lab Exercise No:**36

**Exercise Objective(s):***protected access specifier*

**Exercise:***In the Lab Exercise 35, create another class called City which creates an object for the Car,*

*Truck and Bus class and displays the details through a display () method in the Vehicle class.*

*The other methods and data members should not be accessible by the City class.*

**Solution:**

**package** hsbc.training.day3;

**public** **class** NewRoads

{

**public** **static** **void** main(String[] args)

{

City c=**new** City();

c.city();

}

}

//class Vehicle is the base/super class

**class** Vehicle

{

**protected** String color,brand;

**protected** **int** regNo,noOfWheels;

//Parameterized constructor of Vehicle class

Vehicle(**int** regNo,String brand,**int** noOfWheels,String color)

{

**this**.regNo=regNo;

**this**.brand=brand;

**this**.noOfWheels=noOfWheels;

**this**.color=color;

}

//display vehicle details

**protected** **void** displayDetails()

{

System.***out***.println("Registration Number ==>"+regNo);

System.***out***.println("Brand Name ==>"+brand);

System.***out***.println("Color ==>"+color);

System.***out***.println("No. of Wheels ==>"+noOfWheels);

}

}

//subclass of Vehicle

**class** Truck **extends** Vehicle

{

//Truck constructor calling Vehicle Constructor

Truck(String owner,**int** regNo, String brand, **int** noOfWheels, String color)

{

**super**(regNo, brand, noOfWheels, color);

}

}

//subclass of Vehilce

**class** Bus **extends** Vehicle

{

//Bus constructor calling Vehicle Constructor

Bus(String owner,**int** regNo, String brand, **int** noOfWheels, String color)

{

**super**(regNo, brand, noOfWheels, color);

}

}

//subclass of Vehilce

**class** Car **extends** Vehicle

{

//Car constructor calling Vehicle Constructor

Car(String owner,**int** regNo, String brand, **int** noOfWheels, String color)

{

**super**(regNo, brand, noOfWheels, color);

}

}

//subclass of Vehilce

**class** Bike **extends** Vehicle

{

//Bike constructor calling Vehicle Constructor

Bike(String owner,**int** regNo, String brand, **int** noOfWheels, String color)

{

**super**(regNo, brand, noOfWheels, color);

}

}

**class** City

{

**public** **void** city()

{

//creating Truck object

Truck t1=**new** Truck("Tom",12345,"Mahindra",6,"Orange");

t1.displayDetails();

//creating Bus object

Bus b=**new** Bus("Oshia",54321,"Tata Motors",4,"Red");

b.displayDetails();

//creating Car object

Car c=**new** Car("Lucas",9876,"Audi",4,"Black");

c.displayDetails();

//creating Bike object

Bike bk=**new** Bike("Rachel",56789,"Honda",2,"Red");

bk.displayDetails();

}

}

**Lab Exercise No:**37

**Exercise Objective(s):***Overriding*

**Exercise:** *In the Lab Exercise 30, create a super class called Animal and make all the existing classes as the sub class for Animal class. Move the method isVegetarian and canClimb to the super class*

*and implement generically. Whenever necessary change the implementation of these methods*

*in the respective subclasses. Display the characteristic of each animal.*

**Solution:**

**Lab Exercise No:**38

**Exercise Objective(s):***final Keyword*

**Exercise:***In the Lab Exercise 37, make the Lion, Tiger, Deer, Monkey, Elephant and Giraffe classes such that these classes cannot be inherited.*

**Solution:**

**Lab Exercise No:**39

**Exercise Objective(s):***Polymorphism*

**Exercise:** *Create a class called Worker. Write classes DailyWorker and SalariedWorker that inherit from Worker.Every worker has a name and a salaryrate. Write method Pay (int hours) to compute*

*the week pay of every worker. A Daily worker is paid on the basis of the number of days*

*she/he works.The salaried worker gets paid the wage for 40 hours a week no matter what the*

*actual hours are. Test this program to calculate the pay of workers.*

**Solution:**

**package** hsbc.training.day3;

//Main Class Calculation

**public** **class** PayCalculation

{

**public** **static** **void** main(String[] args)

{

DailyWorker dw=**new** DailyWorker("Ross",6000);

dw.pay(36);

SalariedWorker sw=**new** SalariedWorker("Rachel",8000);

sw.pay();

}

}

//Class Worker is the base class of class daily worker and salaried worker

**class** Worker

{

**protected** String name;

**protected** **int** salaryRate;

//parameterized constructor of class worker

Worker(String nm,**int** sal)

{

name=nm;

salaryRate=sal;

}

//displays the details of workers

**protected** **void** display()

{

System.***out***.println("Name ==>"+name );

System.***out***.println("Salary Rate ==> "+salaryRate);

}

}

//Class DailyWorker is the subclass of class Worker

**class** DailyWorker **extends** Worker

{

**int** salary;

DailyWorker(String nm, **int** sal)

{

//calling the constructor of base class worker

**super**(nm, sal);

}

//calculates the weekly salary on the basis of working hours and salary rate

**public** **void** pay(**int** hrs)

{

salary=hrs\*salaryRate;

System.***out***.println("--------- Daily worker Details --------");

display();

System.***out***.println("Salary ==> "+salary);

}

}

//Class SalariedWorker is the subclass of class Worker

**class** SalariedWorker **extends** Worker

{

**int** salary;

**final** **int** hours=40;

SalariedWorker(String nm, **int** sal)

{

**super**(nm, sal);

}

//calculates the weekly salary on the basis of working hours and salary rate

//weekly working hours for salaried worker is fixed i.e 40;

**public** **void** pay()

{

salary=hours\*salaryRate;

System.***out***.println("--------- Salaried worker Details --------");

display();

System.***out***.println("Salary ==> "+salary);

}

}

**Lab Exercise No:**40

**Exercise Objective(s):***Polymorphism*

**Exercise:***ModifyLab Exercise 38 by creating intermediate Super classes called WildAnimals and*

*DomesticAnimals and create corresponding subclasses for the same. Create generic*

*methods in the super class and display the details of wild animals, domestic animals in*

*general separately and also display the details of each animal separately.*

**Recommended duration:** *20 Mins*

**Solution Guidance (if applicable):** *NA*

**Lab Exercise No:**41

**Exercise Objective(s):***abstract classes*

**Exercise:***Create a class called Shape3D with the following method signatures alone, volume () and*

*surfaceArea (). Then create subclasses like Cylinder, Sphere, and Cubeetc and implement*

*these methods.*

**Solution:**

**package** hsbc.training.day3;

**public** **class** Shapes

{

**public** **static** **void** main(String args[])

{

Cylinder c=**new** Cylinder(4,5.5);

c.display();

Sphere s=**new** Sphere(5.5);

s.display();

Cube cb=**new** Cube(4);

cb.display();

}

}

//Class Shape3D declared as abstract

**abstract** **class** Shape3D

{

**public** **abstract** **double** volume();

**public** **abstract** **double** surfaceArea();

**protected** **final** **float** pi=3.14f;

}

//class Cylinder using methods of abstract Shape3D

**class** Cylinder **extends** Shape3D

{

**double** radius,height;

//parameterized constructor to accept radius and height

Cylinder(**double** r, **double** h)

{

radius=r;

height=h;

}

//calculate volume of cylinder

@Override

**public** **double** volume()

{

**double** vol=pi\*Math.*pow*(radius, 2)\*height;

**return** vol;

}

//calculate surface area of cylinder

@Override

**public** **double** surfaceArea()

{

**double** surArea=2\*pi\*radius\*(radius+height);

**return** surArea;

}

//display volume and surface area of cylinder

**public** **void** display()

{

System.***out***.println("Volume of Cylinder ==>"+volume());

System.***out***.println("Surface Area of Cylinder ==>"+surfaceArea());

}

}

//class Sphere using methods of abstract Shape3D

**class** Sphere **extends** Shape3D

{

**double** radius;

//parameterized constructor to accept radius

Sphere(**double** r)

{

radius=r;

}

//calculate volume of sphere

@Override

**public** **double** volume()

{

**float** no=(**float**)4/3;

**double** vol=no\*pi\*Math.*pow*(radius,3);

**return** vol;

}

//calculate surfaceArea of sphere

@Override

**public** **double** surfaceArea()

{

**double** surArea=4\*pi\*Math.*pow*(radius,2);

**return** surArea;

}

//display volume and surface area of sphere

**public** **void** display()

{

System.***out***.println("Volume of Sphere ==>"+volume());

System.***out***.println("Surface Area of Sphere ==>"+surfaceArea());

}

}

//class Cube using methods of abstract Shape3D

**class** Cube **extends** Shape3D

{

**double** a;

//parameterized constructor to accept edge

Cube(**double** x)

{

a=x;

}

//calculate volume of sphere

@Override

**public** **double** volume()

{

**double** vol=Math.*pow*(a,3);

**return** vol;

}

//calculate surfaceArea of sphere

@Override

**public** **double** surfaceArea()

{

**double** surArea=6\*Math.*pow*(a,2);

**return** surArea;

}

//display volume and surface area of Cube

**public** **void** display()

{

System.***out***.println("Volume of Cube ==>"+volume());

System.***out***.println("Surface Area of Cube ==>"+surfaceArea());

}

}

**Lab Exercise No:**42

**Exercise Objective(s):***abstract classes*

**Exercise:***Create the classes required to store data regarding different types of courses that employees Ina company can enroll for. All courses have name and course fee. Courses are also either*

*classroom delivered or delivered online. Courses could also be full time or part time. The*

*program must be menu based input which enables the course coordinator to register*

*employees for courses, list out employees registered for specific courses, deregister employees*

*from a course.*

**Recommended duration:***30Mins*

**Solution Guidance (if applicable):***The output can be like this,*

*Enter the number of the choice you want to choose:*

1. *To register for a course*
2. *To deregister for a course*
3. *To list the courses offered*
4. *To list the employees registered for a specific course.*

**JAVA**

**Interface**

**Lab Exercise No:**43

**Exercise Objective(s):***Implementation of an interface*

**Exercise:***Implement Lab Exercise 41using Interfaces.*

**Solution :**

**package** hsbc.training.day3;

**public** **class** Shapes

{

**public** **static** **void** main(String args[])

{

Cylinder c=**new** Cylinder(4,5.5);

c.display();

Sphere s=**new** Sphere(5.5);

s.display();

Cube cb=**new** Cube(4);

cb.display();

}

}

//Class Shape3D declared as abstract

**interface** Shape3D

{

**public** **double** volume();

**public** **double** surfaceArea();

**public** **void** display();

**float** ***pi***=3.14f;

}

//class Cylinder using methods of interface Shape3D

**class** Cylinder **implements** Shape3D

{

**double** radius,height;

//parameterized constructor to accept radius and height

Cylinder(**double** r, **double** h)

{

radius=r;

height=h;

}

//calculate volume of cylinder

@Override

**public** **double** volume()

{

**double** vol=***pi***\*Math.*pow*(radius, 2)\*height;

**return** vol;

}

//calculate surface area of cylinder

@Override

**public** **double** surfaceArea()

{

**double** surArea=2\****pi***\*radius\*(radius+height);

**return** surArea;

}

//display volume and surface area of cylinder

**public** **void** display()

{

System.***out***.println("Volume of Cylinder ==>"+volume());

System.***out***.println("Surface Area of Cylinder ==>"+surfaceArea());

}

}

//class Sphere using methods of interface Shape3D

**class** Sphere **implements** Shape3D

{

**double** radius;

//parameterized constructor to accept radius

Sphere(**double** r)

{

radius=r;

}

//calculate volume of sphere

@Override

**public** **double** volume()

{

**float** no=(**float**)4/3;

**double** vol=no\****pi***\*Math.*pow*(radius,3);

**return** vol;

}

//calculate surfaceArea of sphere

@Override

**public** **double** surfaceArea()

{

**double** surArea=4\****pi***\*Math.*pow*(radius,2);

**return** surArea;

}

//display volume and surface area of sphere

**public** **void** display()

{

System.***out***.println("Volume of Sphere ==>"+volume());

System.***out***.println("Surface Area of Sphere ==>"+surfaceArea());

}

}

//class Cube using methods of interface Shape3D

**class** Cube **implements** Shape3D

{

**double** a;

//parameterized constructor to accept edge of cube

Cube(**double** x)

{

a=x;

}

//calculate volume of sphere

@Override

**public** **double** volume()

{

**double** vol=Math.*pow*(a,3);

**return** vol;

}

//calculate surfaceArea of sphere

@Override

**public** **double** surfaceArea()

{

**double** surArea=6\*Math.*pow*(a,2);

**return** surArea;

}

//display volume and surface area of Cube

**public** **void** display()

{

System.***out***.println("Volume of Cube ==>"+volume());

System.***out***.println("Surface Area of Cube ==>"+surfaceArea());

}

}

**Lab Exercise No:**44

**Exercise Objective(s):***Implementing more than one interface.*

**Exercise:***Create two interfaces namely Drawable and Fillable. Create class called Line, Circle, Square and implement following methods through interface.*

*<I>Drawable ------ drawingColor(), thickness()*

*<I>Fillable ---------- fillingColor(), size()*

**Soluton:**

**package** hsbc.training.day3;

**public** **class** Draw

{

**public** **static** **void** main(String[] args)

{

Line l=**new** Line();

l.display();

Square s=**new** Square();

s.display();

Circle c=**new** Circle();

c.display();

}

}

//interface drawable with method drawingColor,thickness

**interface** Drawable

{

**public** **void** drawingColor();

**public** **void** thickness();

}

//interface Fillable with method fillingColor,size

**interface** Fillable

{

**public** **void** fillingColor();

**public** **void** size();

}

//class line implements the interface Drawable,Fillable

**class** Line **implements** Drawable,Fillable

{

//overrided method of interface Fillable

@Override

**public** **void** fillingColor()

{

System.***out***.println(" Filling color ==> Blue");

}

//overrided method of interface Fillable

@Override

**public** **void** size()

{

System.***out***.println(" Size ==> 64 cm");

}

//overrided method of interface Drawable

@Override

**public** **void** drawingColor()

{

System.***out***.println(" Drawing color ==> Black");

}

//overrided method of interface Drawable

@Override

**public** **void** thickness()

{

System.***out***.println(" Thickness ==> 40");

}

//display the details

**public** **void** display()

{

System.***out***.println("----------------Line Details-----------------");

fillingColor();

size();

drawingColor();

thickness();

}

}

//class Square implements the interface Drawable,Fillable

**class** Square **implements** Drawable,Fillable

{

//overrided method of interface Fillable

@Override

**public** **void** fillingColor()

{

System.***out***.println(" Filling color ==> Yellow");

}

//overrided method of interface Fillable

@Override

**public** **void** size()

{

System.***out***.println(" Size ==> 87");

}

//overrided method of interface Drawable

@Override

**public** **void** drawingColor()

{

System.***out***.println(" Drawing color ==> Red");

}

//overrided method of interface Drawable

@Override

**public** **void** thickness()

{

System.***out***.println(" Thickness ==> 50");

}

//display the details

**public** **void** display()

{

System.***out***.println("-------------Square Details-----------------");

fillingColor();

size();

drawingColor();

thickness();

}

}

//class Circle implements the interface Drawable,Fillable

**class** Circle **implements** Drawable,Fillable

{

//overrided method of interface Fillable

@Override

**public** **void** fillingColor()

{

System.***out***.println(" Filling color ==> Green");

}

//overrided method of interface Fillable

@Override

**public** **void** size()

{

System.***out***.println(" Size ==> 64");

}

//overrided method of interface Drawable

@Override

**public** **void** drawingColor()

{

System.***out***.println(" Drawing color ==> Purple");

}

//overrided method of interface Drawable

@Override

**public** **void** thickness()

{

System.***out***.println(" Thickness ==> 40");

}

//display the details

**public** **void** display()

{

System.***out***.println("--------------Circle Details-----------------");

fillingColor();

size();

drawingColor();

thickness();

}

}

**Lab Exercise No:**45

**Exercise Objective(s):***Implementation of an interface*

**Exercise:***In Lab Exercise 39, create a package called finance and within it create an interface called*

*Payable.It should define the getPayInfo() method that all the worker classes will implement.*

*Now display the details of the monthly pay of the workers.*

**Solution:**

**package** finance;

//interfacePayable

**public** **interface** Payable

{

**public** **void** getPayInfo(**int** hrs);

}

**package** hsbc.training.day3;

**import** finance.Payable;

**public** **class** PayCalculation

{

**public** **static** **void** main(String[] args)

{

DailyWorker dw=**new** DailyWorker("Ross",100);

dw.pay(36);

SalariedWorker sw=**new** SalariedWorker("Rachel",240);

sw.pay();

}

}

//Class Worker is the base class of class daily worker and salaried worker

**class** Worker **implements** Payable

{

**protected** String name;

**protected** **int** salaryRate;

//parameterized constructor of class worker

Worker(String nm,**int** sal)

{

name=nm;

salaryRate=sal;

}

//calculate monthly salary and displays the details of workers

@Override

**public** **void** getPayInfo(**int** hrs)

{

**double** sal=hrs\*salaryRate;

System.***out***.println("Name ==>"+name );

System.***out***.println("Hourly Salary Rate ==> "+salaryRate);

System.***out***.println("Monthly Rate ==> "+sal);

}

}

//Class DailyWorker is the subclass of class Worker

**class** DailyWorker **extends** Worker

{

**int** salary;

DailyWorker(String nm, **int** sal)

{

//calling the constructor of base class worker

**super**(nm, sal);

}

//calculates the weekly salary on the basis of working hours and salary rate

//displays weekly and monthly salary

**public** **void** pay(**int** hrs)

{

salary=hrs\*salaryRate;

System.***out***.println("--------- Daily worker Details --------");

//calculating daily working hours

**int** perDayHrs=hrs/7;

//calculating monthly working hours

**int** monthlyHrs=perDayHrs\*30;

getPayInfo(monthlyHrs);

System.***out***.println("Weekly Salary ==> "+salary);

}

}

//Class SalariedWorker is the subclass of class Worker

**class** SalariedWorker **extends** Worker

{

**int** salary;

**final** **int** hours=40;

SalariedWorker(String nm, **int** sal)

{

**super**(nm, sal);

}

//calculates the weekly salary on the basis of working hours and salary rate

//weekly working hours for salaried worker is fixed i.e 40;

//displays weekly and monthly salary

**public** **void** pay()

{

salary=hours\*salaryRate;

System.***out***.println("--------- Salaried worker Details --------");

//calculating daily working hours

**int** perDayHrs=hours/7;

//calculating monthly working hours

**int** monthlyHrs=perDayHrs\*30;

getPayInfo(monthlyHrs);

System.***out***.println("Weekly Salary ==> "+salary);

}

}

**Lab Exercise No:**46

**Exercise Objective(s):***Implementation of an interface*

**Exercise:***Createa package called bank with the following Interfaces.*

*<Interface> Account*

*<Interface>DepositAcc <Interface>LoanAcc*

*<Interface> Interest*

*<Interface>CreditInterest <Interface>DebitInterest*

1. *<Interface> Account*

*Data members: Four String variables to hold the account type “Savings, Fixed,PersonalLoan,*

*HousingLoan”*

*Methods: createAcc()*

1. *<Interface>DepositAcc*

*Methods: withdraw (), deposit(),getBalance()*

1. *<Interface>LoanAcc*

*Methods: repayPrincipal (),payInterest (),payPartialPrincipal ()*

1. *<Interface>Interest*

*Data members: Four double variables to hold the interest percentage of Savings account, Fixed*

*deposit account,PersonalLoan account and HousingLoan account.*

*Methods: calcInt()*

1. *<Interface>CreditInterest*

*Methods: addMonthlyInt(),addHalfYrlyInt(),addAnnualInt()*

1. *<Interface>DebitInterest*

*Methods: deductMonthlyInt(),deductHalfYrlyInt(),deductAnnualInt()*

*Create a package called BankImpl and create the following classes in it.*

1. *SavingsAcc which implements DepositAcc and CreditInterest*
2. *FDAcc which implements DepositAcc and CreditInterest*
3. *PersonalLoanAcc which implements LoanAcc and DebitInterest*
4. *HousingLoanAcc which implements LoanAcc and DebitInterest*

*Now create a class called MyAccount and create instances of all the accounts and generate appropriate output.*

***Solution:***

**package** Bank;

// interface Account

**public** **interface** Account

{

String ***savings***="Savings Account";

String ***fixed*** = "Fixed Account";

String ***personalLoan*** = "Personal Loan";

String ***housingLoan*** = "Housing Loan";

**public** **void** createAcc();

}

// interface Interest

**interface** Interest

{

**double** ***savingsAccount*** = 34.5;

**double** ***fixedDepositAccount***=33.44;

**double** ***personalLoanAccount***=26.55;

**double** ***housingLoanAccount***=22.443;

**public** **void** calcInt();

}

**package** Bank;

//interface CreditInterest extends interface Interest

**public** **interface** CreditInterest **extends** Interest

{

**public** **void** addMonthlyInt();

**public** **void** addHalfYrlyInt();

**public** **void** addAnnualInt();

}

**package** Bank;

//interface CreditInterest extends interface Interest

**public** **interface** DebitInterest **extends** Interest

{

**public** **void** deductMonthlyInt();

**public** **void** deductHalfYrlyInt();

**public** **void** deductAnnualInt();

}

**package** Bank;

// interface LoanAcc extends interface Account

**public** **interface** LoanAcc **extends** Account

{

**public** **void** repayPrincipal();

**public** **void** payInterest();

**public** **void** payPartialPrincipal();

}

**package** Bank;

//interface DepositAcc extends interface Account

**public** **interface** DepositAcc **extends** Account

{

**public** **void** withdraw ();

**public** **void** deposit();

**public** **void** getBalance();

}

**package** BankImpl;

**import** java.util.Scanner;

**import** Bank.Account;

**import** Bank.CreditInterest;

**import** Bank.DebitInterest;

**import** Bank.DepositAcc;

**import** Bank.LoanAcc;

**public** **class** MyAccounts

{

**public** **static** **void** main(String[] args)

{

SavingsAcc s=**new** SavingsAcc();

s.display();

FDAcc f=**new** FDAcc();

f.display();

PersonalLoanAcc p=**new** PersonalLoanAcc();

p.display();

HousingLoanAcc h=**new** HousingLoanAcc();

h.display();

}

}

**class** SavingsAcc **implements** DepositAcc,CreditInterest

{

Scanner s=**new** Scanner(System.***in***);

**long** accNo;

**double** deposit;

**double** balance;

**double** withdraw;

**double** monthlyInterestRate;

**double** annualInterestRate;

**double** halfYearlyInterestRate;

**double** interest=***savingsAccount***;

//creates account and accepts account no and balance amount

@Override

**public** **void** createAcc()

{

System.***out***.println("Create a "+***savings***);

System.***out***.println("Ente the account number ==>");

accNo=s.nextLong();

System.***out***.println("Ente the account opening balance ==>");

balance=s.nextLong();

}

//calculate interest

@Override

**public** **void** calcInt()

{

interest = Math.*round*(interest \* 100.0) / 100.0;

System.***out***.println("Interest Rate for saving accounts is "+interest);

}

//calculate monthlyInterestRate

@Override

**public** **void** addMonthlyInt()

{

monthlyInterestRate = annualInterestRate / 12;

balance=balance+monthlyInterestRate;

System.***out***.println("Monthly Interest Rate ==>"+monthlyInterestRate);

System.***out***.println("Balance After Monthly Interest Rate ==>"+balance);

}

//calculate halfYearlyInterestRate

@Override

**public** **void** addHalfYrlyInt()

{

halfYearlyInterestRate =annualInterestRate / 6;

balance=balance+halfYearlyInterestRate;

System.***out***.println("Half Yearly Interest Rate ==>"+halfYearlyInterestRate);

System.***out***.println("Balance After Half Yearly Interest Rate ==>"+balance);

}

//calculate halfYearlyInterestRate

@Override

**public** **void** addAnnualInt()

{

annualInterestRate = interest;

balance=balance+annualInterestRate;

System.***out***.println("Annual Interest Rate ==>"+annualInterestRate);

System.***out***.println("Balance After Yearly Interest Rate ==>"+balance);

}

//method to withdraw amount from account

@Override

**public** **void** withdraw()

{

//accepting the amount to be withdrawn

System.***out***.println("Enter the withdrawal amount ==>");

withdraw=s.nextDouble();

//checking whether the amount is available in the account

**if**(balance>withdraw)

{

//if yes then deduct the amount from main balance

balance=balance-withdraw;

}

**else**

{

//if not the display this message

System.***out***.println("Withdrawal decline !! No enough balance in the account.");

}

}

//method to deposit balance in bank account

@Override

**public** **void** deposit()

{

//accepting deposit amount

System.***out***.println("Enter the deposit amount ==>");

deposit=s.nextDouble();

//adding the accepted amount to the balance

balance=balance+deposit;

}

//method to get the account balance

@Override

**public** **void** getBalance()

{

System.***out***.println("Account Balance ==>"+balance);

}

**public** **void** display()

{

createAcc();

calcInt();

addAnnualInt();

addMonthlyInt();

addHalfYrlyInt();

deposit();

getBalance();

}

}

**class** FDAcc **implements** DepositAcc,CreditInterest

{

Scanner s=**new** Scanner(System.***in***);

**long** accNo;

**double** deposit;

**double** balance;

**double** withdraw;

**double** monthlyInterestRate;

**double** annualInterestRate;

**double** halfYearlyInterestRate;

**double** interest=***fixedDepositAccount***;

//creates account and accepts account no and balance amount

@Override

**public** **void** createAcc()

{

System.***out***.println("Create a "+***fixed***);

System.***out***.println("Ente the account number ==>");

accNo=s.nextLong();

System.***out***.println("Ente the account opening balance ==>");

balance=s.nextLong();

}

//calculate interest

@Override

**public** **void** calcInt()

{

interest = Math.*round*(interest \* 100.0) / 100.0;

System.***out***.println("Interest Rate for fixed deposit accounts is "+interest);

}

//calculate monthlyInterestRate

@Override

**public** **void** addMonthlyInt()

{

monthlyInterestRate = annualInterestRate / 12;

balance=balance+monthlyInterestRate;

System.***out***.println("Monthly Interest Rate ==>"+monthlyInterestRate);

System.***out***.println("Balance After Monthly Interest Rate ==>"+balance);

}

//calculate halfYearlyInterestRate

@Override

**public** **void** addHalfYrlyInt()

{

halfYearlyInterestRate =annualInterestRate / 6;

balance=balance+halfYearlyInterestRate;

System.***out***.println("Half Yearly Interest Rate ==>"+halfYearlyInterestRate);

System.***out***.println("Balance After Half Yearly Interest Rate ==>"+balance);

}

//calculate halfYearlyInterestRate

@Override

**public** **void** addAnnualInt()

{

annualInterestRate = interest;

balance=balance+annualInterestRate;

System.***out***.println("Annual Interest Rate ==>"+annualInterestRate);

System.***out***.println("Balance After Annual Interest Rate ==>"+balance);

}

//method to withdraw amount from account

@Override

**public** **void** withdraw()

{

//accepting the amount to be withdrawn

System.***out***.println("Enter the withdrawal amount ==>");

withdraw=s.nextDouble();

//checking whether the amount is available in the account

**if**(balance>withdraw)

{

//if yes then deduct the amount from main balance

balance=balance-withdraw;

}

**else**

{

//if not the display this message

System.***out***.println("Withdrawal decline !! No enough balance in the account.");

}

}

//method to deposit balance in bank account

@Override

**public** **void** deposit()

{

//accepting deposit amount

System.***out***.println("Enter the deposit amount ==>");

deposit=s.nextDouble();

//adding the accepted amount to the balance

balance=balance+deposit;

}

//method to get the account balance

@Override

**public** **void** getBalance()

{

System.***out***.println("Account Balance ==>"+balance);

}

**public** **void** display()

{

createAcc();

calcInt();

addAnnualInt();

addMonthlyInt();

addHalfYrlyInt();

deposit();

getBalance();

}

}

**class** PersonalLoanAcc **implements** LoanAcc,DebitInterest

{

Scanner s=**new** Scanner(System.***in***);

**long** accNo;

**double** deposit;

**double** balance;

**double** withdraw;

**double** monthlyInterestRate;

**double** annualInterestRate;

**double** halfYearlyInterestRate;

**double** interest=***personalLoanAccount***;

//creates account and accepts account no and balance amount

@Override

**public** **void** createAcc()

{

System.***out***.println("Create a "+***personalLoan***);

System.***out***.println("Ente the account number ==>");

accNo=s.nextLong();

System.***out***.println("Ente the account opening balance ==>");

balance=s.nextLong();

}

//calculate interest

@Override

**public** **void** calcInt()

{

interest = Math.*round*(interest \* 100.0) / 100.0;

System.***out***.println("Interest Rate for personal loan is "+interest);

}

//calculate deductMonthlyInterestRate

@Override

**public** **void** deductMonthlyInt()

{

monthlyInterestRate = annualInterestRate / 12;

System.***out***.println("Monthly Interest Rate ==>"+monthlyInterestRate);

//checking whether the amount is available in the account

**if**(balance>monthlyInterestRate)

{

//if yes then deduct the amount from main balance

balance=balance-monthlyInterestRate;

System.***out***.println("Balance After Monthly Interest Rate ==>"+balance);

}

**else**

{

//if not the display this message

System.***out***.println("Withdrawal decline !! No enough balance in the account.");

}

}

//calculate deductHalfYearlyInterestRate

@Override

**public** **void** deductHalfYrlyInt()

{

halfYearlyInterestRate = annualInterestRate / 6;

System.***out***.println("Hal Yearly Interest Rate ==>"+monthlyInterestRate);

//checking whether the amount is available in the account

**if**(balance>halfYearlyInterestRate)

{

//if yes then deduct the amount from main balance

balance=balance-halfYearlyInterestRate;

System.***out***.println("Balance After Monthly Interest Rate ==>"+balance);

}

**else**

{

//if not the display this message

System.***out***.println("Withdrawal decline !! No enough balance in the account.");

}

}

//calculate deductAnnualInterestRate

@Override

**public** **void** deductAnnualInt()

{

annualInterestRate = interest;

System.***out***.println("Annual Interest Rate ==>"+annualInterestRate);

//checking whether the amount is available in the account

**if**(balance>annualInterestRate)

{

//if yes then deduct the amount from main balance

balance=balance-monthlyInterestRate;

System.***out***.println("Balance After Monthly Interest Rate ==>"+balance);

}

**else**

{

//if not the display this message

System.***out***.println("Withdrawal decline !! No enough balance in the account.");

}

}

@Override

**public** **void** repayPrincipal()

{

System.***out***.println("Repay Principal");

}

@Override

**public** **void** payInterest()

{

System.***out***.println("Pay Interest");

}

@Override

**public** **void** payPartialPrincipal()

{

System.***out***.println("Pay Partial Principal");

}

**public** **void** display()

{

createAcc();

calcInt();

deductAnnualInt();

deductMonthlyInt();

deductHalfYrlyInt();

repayPrincipal();

payInterest();

payPartialPrincipal();

}

}

**class** HousingLoanAcc **implements** LoanAcc,DebitInterest

{

Scanner s=**new** Scanner(System.***in***);

**long** accNo;

**double** deposit;

**double** balance;

**double** withdraw;

**double** monthlyInterestRate;

**double** annualInterestRate;

**double** halfYearlyInterestRate;

**double** interest=***housingLoanAccount***;

//creates account and accepts account no and balance amount

@Override

**public** **void** createAcc()

{

System.***out***.println("Create a "+***housingLoan***);

System.***out***.println("Ente the account number ==>");

accNo=s.nextLong();

System.***out***.println("Ente the account opening balance ==>");

balance=s.nextLong();

}

//calculate interest

@Override

**public** **void** calcInt()

{

interest = Math.*round*(interest \* 100.0) / 100.0;

System.***out***.println("Interest Rate for personal loan is "+interest);

}

//calculate deductMonthlyInterestRate

@Override

**public** **void** deductMonthlyInt()

{

monthlyInterestRate = annualInterestRate / 12;

System.***out***.println("Monthly Interest Rate ==>"+monthlyInterestRate);

//checking whether the amount is available in the account

**if**(balance>monthlyInterestRate)

{

//if yes then deduct the amount from main balance

balance=balance-monthlyInterestRate;

System.***out***.println("Balance After Monthly Interest Rate ==>"+balance);

}

**else**

{

//if not the display this message

System.***out***.println("Withdrawal decline !! No enough balance in the account.");

}

}

//calculate deductHalfYearlyInterestRate

@Override

**public** **void** deductHalfYrlyInt()

{

halfYearlyInterestRate = annualInterestRate / 6;

System.***out***.println("Hal Yearly Interest Rate ==>"+monthlyInterestRate);

//checking whether the amount is available in the account

**if**(balance>halfYearlyInterestRate)

{

//if yes then deduct the amount from main balance

balance=balance-halfYearlyInterestRate;

System.***out***.println("Balance After Monthly Interest Rate ==>"+balance);

}

**else**

{

//if not the display this message

System.***out***.println("Withdrawal decline !! No enough balance in the account.");

}

}

//calculate deductAnnualInterestRate

@Override

**public** **void** deductAnnualInt()

{

annualInterestRate = interest;

System.***out***.println("Annual Interest Rate ==>"+annualInterestRate);

//checking whether the amount is available in the account

**if**(balance>annualInterestRate)

{

//if yes then deduct the amount from main balance

balance=balance-monthlyInterestRate;

System.***out***.println("Balance After Monthly Interest Rate ==>"+balance);

}

**else**

{

//if not the display this message

System.***out***.println("Withdrawal decline !! No enough balance in the account.");

}

}

@Override

**public** **void** repayPrincipal()

{

System.***out***.println("Repay Principal");

}

@Override

**public** **void** payInterest()

{

System.***out***.println("Pay Interest");

}

@Override

**public** **void** payPartialPrincipal()

{

System.***out***.println("Pay Partial Principal");

}

**public** **void** display()

{

createAcc();

calcInt();

deductAnnualInt();

deductMonthlyInt();

deductHalfYrlyInt();

repayPrincipal();

payInterest();

payPartialPrincipal();

}

}

**JAVA**

**Inner class**

**Lab Exercise No:**47

**Exercise Objective(s):***Inner classes*

**Exercise:***Create a class called BankAccount with deposit (), withdraw () and getBalance () methods.*

*Create an inner class called InterestAdder and implement the interest calculations and add*

*the interest to the current balance.*

**Solution:**

**package** hsbc.training.day3;

**import** java.util.Scanner;

**public** **class** BankAccountMain

{

**public** **static** **void** main(String[] args)

{

//creating the object of outer class BankAccount

BankAccount b=**new** BankAccount("Rasi",8765423,9000);

//creating the object of inner class InterestAdder

BankAccount.InterestAdder i=b.**new** InterestAdder();

//calling the deposit method of BankAccount

b.deposit();

//calling the withdraw method of BankAccount

b.withdraw();

//calling the getBalance method of BankAccount

b.getBalance();

//calling the interestBal method of InterestAdder

i.interestBal();

}

}

//outer class BankAccount

**class** BankAccount

{

String name;

**long** accountNo;

**double** bal=0;

Scanner s=**new** Scanner(System.***in***);

//parameterized constructor BankAccount with name,account no and amount

BankAccount(String nm,**long** accNo,**double** amt)

{

name=nm;

accountNo=accNo;

bal=amt;

}

//method to deposit balance in bank account

**public** **void** deposit()

{

**double** dep;

//accepting deposit amount

System.***out***.println("Enter the deposit amount ==>");

dep=s.nextDouble();

//adding the accepted amount to the balance

bal=bal+dep;

}

//method to withdraw amount from account

**public** **void** withdraw()

{

**double** wiDraw;

//accepting the amount to be withdrawn

System.***out***.println("Enter the withdrawal amount ==>");

wiDraw=s.nextDouble();

//checking whether the amount is available in the account

**if**(bal>wiDraw)

{

//if yes then deduct the amount from main balance

bal=bal-wiDraw;

}

**else**

{

//if not the display this message

System.***out***.println("Withdrawal decline !! No enough balance in the account.");

}

}

//method to get the account balance

**public** **void** getBalance()

{

System.***out***.println("Account Balance ==>"+bal);

}

//inner class InterestAdder

**class** InterestAdder

{

**double** intBal;

//calculates the interest amount and adds it to the balance

**public** **void** interestBal()

{

intBal=bal+(((bal\*12)/100)/12);

System.***out***.println("Account Balance after adding interest ==>"+intBal);

}

}

}

**Lab Exercise No:**48

**Exercise Objective(s):***Local inner classes*

**Exercise:***Create a class called BankAccount with deposit (), withdraw () and getBalance () methods.*

*Create a local inner class inside the getBalance() method called InterestAdder and implement the interest calculations and add the interest to the current balance.*

**Solution:**

**package** hsbc.training.day3;

**import** java.util.Scanner;

**public** **class** BankAccountMain

{

**public** **static** **void** main(String[] args)

{

//creating the object of outer class BankAccount

BankAccount b=**new** BankAccount("Rasi",8765423,9000);

//calling the deposit method of BankAccount

b.deposit();

//calling the withdraw method of BankAccount

b.withdraw();

//calling the getBalance method of BankAccount

b.getBalance();

}

}

//outer class BankAccount

**class** BankAccount

{

String name;

**long** accountNo;

**double** bal=0;

Scanner s=**new** Scanner(System.***in***);

//parameterized constructor BankAccount with name,account no and amount

BankAccount(String nm,**long** accNo,**double** amt)

{

name=nm;

accountNo=accNo;

bal=amt;

}

//method to deposit balance in bank account

**public** **void** deposit()

{

**double** dep;

//accepting deposit amount

System.***out***.println("Enter the deposit amount ==>");

dep=s.nextDouble();

//adding the accepted amount to the balance

bal=bal+dep;

}

//method to withdraw amount from account

**public** **void** withdraw()

{

**double** wiDraw;

//accepting the amount to be withdrawn

System.***out***.println("Enter the withdrawal amount ==>");

wiDraw=s.nextDouble();

//checking whether the amount is available in the account

**if**(bal>wiDraw)

{

//if yes then deduct the amount from main balance

bal=bal-wiDraw;

}

**else**

{

//if not the display this message

System.***out***.println("Withdrawal decline !! No enough balance in the account.");

}

}

//method to get the account balance

**public** **void** getBalance()

{

System.***out***.println("Account Balance ==>"+bal);

//local inner class InterestAdder

**class** InterestAdder

{

**double** intBal;

//calculates the interest amount and adds it to the balance

**public** **void** interestBal()

{

intBal=bal+(((bal\*12)/100)/12);

System.***out***.println("Account Balance after adding interest ==>"+intBal);

}

}

//creating object of local class InterestAdder

InterestAdder ia=**new** InterestAdder();

//calling the interestBal

ia.interestBal();

}

}

**Lab Exercise No:**49

**Exercise Objective(s):***Anonymous inner classes*

**Exercise:***Create a class called BankAccount with deposit (), withdraw () and getBalance () methods.*

*Create an anonymous inner class in the getBalance() method to do the interest calculations*

*and add the interest to the current balance.*

**Solution:**

**package** hsbc.training.day3;

**import** java.util.Scanner;

//abstract class InterestAdder

**abstract** **class** InterestAdder

{

**public** **abstract** **void** interestBal();

}

**public** **class** BankAccountsMain

{

**public** **static** **void** main(String[] args)

{

//creating the object of outer class BankAccount

BankAccounts b=**new** BankAccounts("Rasi",8765423,9000);

//calling the deposit method of BankAccount

b.deposit();

//calling the withdraw method of BankAccount

b.withdraw();

//calling the getBalance method of BankAccount

b.getBalance();

}

}

//outer class BankAccount

**class** BankAccounts

{

String name;

**long** accountNo;

**double** bal=0;

Scanner s=**new** Scanner(System.***in***);

//parameterized constructor BankAccount with name,account no and amount

BankAccounts(String nm,**long** accNo,**double** amt)

{

name=nm;

accountNo=accNo;

bal=amt;

}

//method to deposit balance in bank account

**public** **void** deposit()

{

**double** dep;

//accepting deposit amount

System.***out***.println("Enter the deposit amount ==>");

dep=s.nextDouble();

//adding the accepted amount to the balance

bal=bal+dep;

}

//method to withdraw amount from account

**public** **void** withdraw()

{

**double** wiDraw;

//accepting the amount to be withdrawn

System.***out***.println("Enter the withdrawal amount ==>");

wiDraw=s.nextDouble();

//checking whether the amount is available in the account

**if**(bal>wiDraw)

{

//if yes then deduct the amount from main balance

bal=bal-wiDraw;

}

**else**

{

//if not the display this message

System.***out***.println("Withdrawal decline !! No enough balance in the account.");

}

}

//method to get the account balance

**public** **void** getBalance()

{

System.***out***.println("Account Balance ==>"+bal);

//Anonymous inner class InterestAdder

InterestAdder ia =**new** InterestAdder()

{

**double** intBal;

//calculates the interest amount and adds it to the balance

**public** **void** interestBal()

{

intBal=bal+(((bal\*12)/100)/12);

System.***out***.println("Account Balance after adding interest ==>"+intBal);

}

};

//calling method of Anonymous inner class InterestAdder

ia.interestBal();

}

}