|  |
| --- |
| **Lab08: Polymorphism** |

Designing and implementing Java programs that deal with:

|  |
| --- |
| 1. Polymorphism 2. upcasting and down casting 3. instance of operator 4. Exercise for practice |

|  |
| --- |
| **Exercises** |

Exercise 1 (Department)

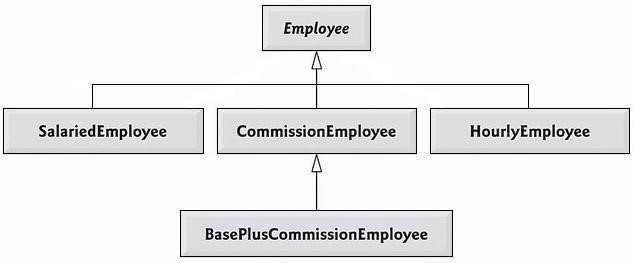
**Exercise 1 (a)**

Create a payroll system using **classes**, **inheritance** and **polymorphism**

Four types of employees paid weekly

1. Salaried employees: fixed salary irrespective of hours
2. Hourly employees: 40 hours salary and overtime (> 40 hours)
3. Commission employees: paid by a percentage of sales
4. Base-plus-commission employees: base salary and a percentage of sales

The information know about each employee is his/her first name, last name and national identity card number. The reset depends on the type of employee.



Step by Step Guidelines

Step 1: Define Employee Class

* Being the base class, Employee class contains the common behavior. Add firstName, lastName and CNIC as attributes of type String
* Provide getter & setters for each attribute
* Write default & parameterized constructors
* Override **toString**() method as shown below

public String **toString**( ) {

return firstName + “ ” + lastName + “ CNIC# ” + CNIC ; }

* Define **earning()** method as shown below public double earnings( )

{ return 0.00;

}

Step 2: Define SalariedEmployee Class

* Extend this class from Employee class.
* Add **weeklySalary** as an attribute of type double
* Provide **getter** & **setters** for this attribute. Make sure that **weeklySalary** never sets to **negative** value. (use if )
* Write **default** & **parameterize** constructor. Don’t forget to call default & parameterize constructors of Employee class.
* Override **toString**() method as shown below

public String toString( ) { return “\nSalaried employee: ” + super.toString(); }

* Override **earning**() method to implement class specific behavior as shown below

public double **earnings**( ) { return weeklySalary;

}

Step 3: Define HourlyEmployee Class

* Extend this class from Employee class.
* Add **wage** and **hours** as attributes of type double
* Provide **getter** & **setters** for these attributes. Make sure that **wage** and **hours** never set to a negative value.
* Write default & parameterize constructor. Don’t forget to call default & parameterize constructors of Employee class.
* Override **toString**() method as shown below public String **toString**( ) {

return “\nHourly employee: ” + super.toString(); }

* Override **earning**() method to implement class specific behaviour as shown below

public double **earnings**( ) { if

(hours <= 40){

return wage \* hours; }

else{

}

Step 4: Define CommissionEmployee Class

* Extend this class form Employee class.
* Add **grossSales** and **commissionRate** as attributes of type double
* Provide **getter** & setters for these attributes. Make sure that grossSales and commissionRate never set to a negative value.
* Write default & parameterize constructor. Don’t forget to call default & parameterize constructors of Employee class.
* Override **toString**() method as shown below public String **toString**( ) { return “\nCommission employee: ” + super.toString();

}

* Override **earning**() method to implement class specific behaviour as shown below

public double **earnings**( ) { return grossSales \* commisionRate;

}

Step 5: Define BasePlusCommissionEmployee Class

* Extend this class form **CommissionEmployee** class not from Employee class. Why? Think on it by yourself
* Add **baseSalary** as an attribute of type double
* Provide **getter** & **setters** for these attributes. Make sure that **baseSalary** never sets to negative value.
* Write default & parameterize constructor. Don’t forget to call default & parameterize constructors of Employee class.
* Override **toString**() method as shown below public String toString( ) { return “\nBase plus Commission employee: ” + super.toString(); }
* Override **earning**() method to implement class specific behaviour as shown below

public double **earnings**( ) { return baseSalary + super.earning(); }

**Source Code:**

**Class Employee:**

public class Employee {

private String firstName;

private String lastName;

private String cnic;

public Employee(String firstName, String lastName, String cnic){

this.firstName= firstName;

this.lastName= lastName;

this.cnic= cnic;

}

public Employee(){}

public String getFirstName(){

return firstName;

}

public void setFirstName(String firstName){

this.firstName= firstName;

}

public String getlastname(){

return lastName;

}

public void setLastName(String lastName){

this.lastName= lastName;

}

public String getCnic(){

return cnic;

}

public void setCnic(String cnic){

this.cnic= cnic;

}

public double earnings( )

{

return 0.00;

}

@Override

public String toString( ) {

return firstName+ " "+ lastName + " CNIC: "+cnic;

}

}

**Class SalariedEmployee:**

public class SalariedEmployee extends Employee {

private double weeklySalary;

public SalariedEmployee(){

}

public SalariedEmployee(String firstName, String lastName, String cnic, double weeklySalary){

super(firstName, lastName, cnic);

if(weeklySalary>=0){

this.weeklySalary= weeklySalary;

}

}

public double getWeeklySalary(){

return weeklySalary;

}

public void setWeeklySalary(double weeklySalary){

if(weeklySalary>=0){

this.weeklySalary= weeklySalary;

}

}

@Override

public String toString() {

return "\nSalaried employee: " + super.toString();

}

@Override

public double earnings() {

return weeklySalary;

}

}

**Class Hourly Employee:**

public class HourlyEmployee extends Employee {

private double wage;

private double hours;

public HourlyEmployee() {

}

public HourlyEmployee(String firstName, String lastName, String CNIC, double wage, double hours) {

super(firstName, lastName, CNIC);

if (wage >= 0) {

this.wage = wage;

}

if (hours >= 0) {

this.hours = hours;

}

}

public double getWage() {

return wage;

}

public void setWage(double wage) {

if (wage >= 0) {

this.wage = wage;

}

}

public double getHours() {

return hours;

}

public void setHours(double hours) {

if (hours >= 0) {

this.hours = hours;

}

}

@Override

public String toString() {

return "\nHourly employee: " + super.toString();

}

@Override

public double earnings() {

if (hours <= 40) {

return wage \* hours;

} else {

return (40 \* wage) + ((hours - 40) \* wage \* 1.5);

}

}

}

**Class CommissionEmployee:**

public class CommissionEmployee extends Employee {

private double grossSales;

private double commissionRate;

public CommissionEmployee() {

}

public CommissionEmployee(String firstName, String lastName, String CNIC, double grossSales, double commissionRate) {

super(firstName, lastName, CNIC);

if (grossSales >= 0) {

this.grossSales = grossSales;

}

if (commissionRate >= 0) {

this.commissionRate = commissionRate;

}

}

public double getGrossSales() {

return grossSales;

}

public void setGrossSales(double grossSales) {

if (grossSales >= 0) {

this.grossSales = grossSales;

}

}

public double getCommissionRate() {

return commissionRate;

}

public void setCommissionRate(double commissionRate) {

if (commissionRate >= 0) {

this.commissionRate = commissionRate;

}

}

@Override

public String toString() {

return "\nCommission employee: " + super.toString();

}

@Override

public double earnings() {

return grossSales \* commissionRate;

}

}

**Class BasePlusCommissionEmployee:**

public class BasePlusCommissionEmployee extends CommissionEmployee{

private double baseSalary;

public BasePlusCommissionEmployee() {

}

public BasePlusCommissionEmployee(String firstName, String lastName, String CNIC, double grossSales, double commissionRate, double baseSalary) {

super(firstName, lastName, CNIC, grossSales, commissionRate);

if (baseSalary >= 0) {

this.baseSalary = baseSalary;

}

}

public double getBaseSalary() {

return baseSalary;

}

public void setBaseSalary(double baseSalary) {

if (baseSalary >= 0) {

this.baseSalary = baseSalary;

}

}

@Override

public String toString() {

return "\nBase plus Commission employee: " + super.toString();

}

@Override

public double earnings() {

return baseSalary + super.earnings();

}

}

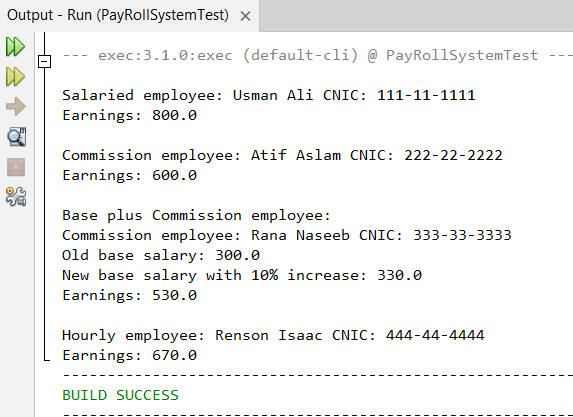
**Exercise 1 (b)**

Step 6: Putting it all Together

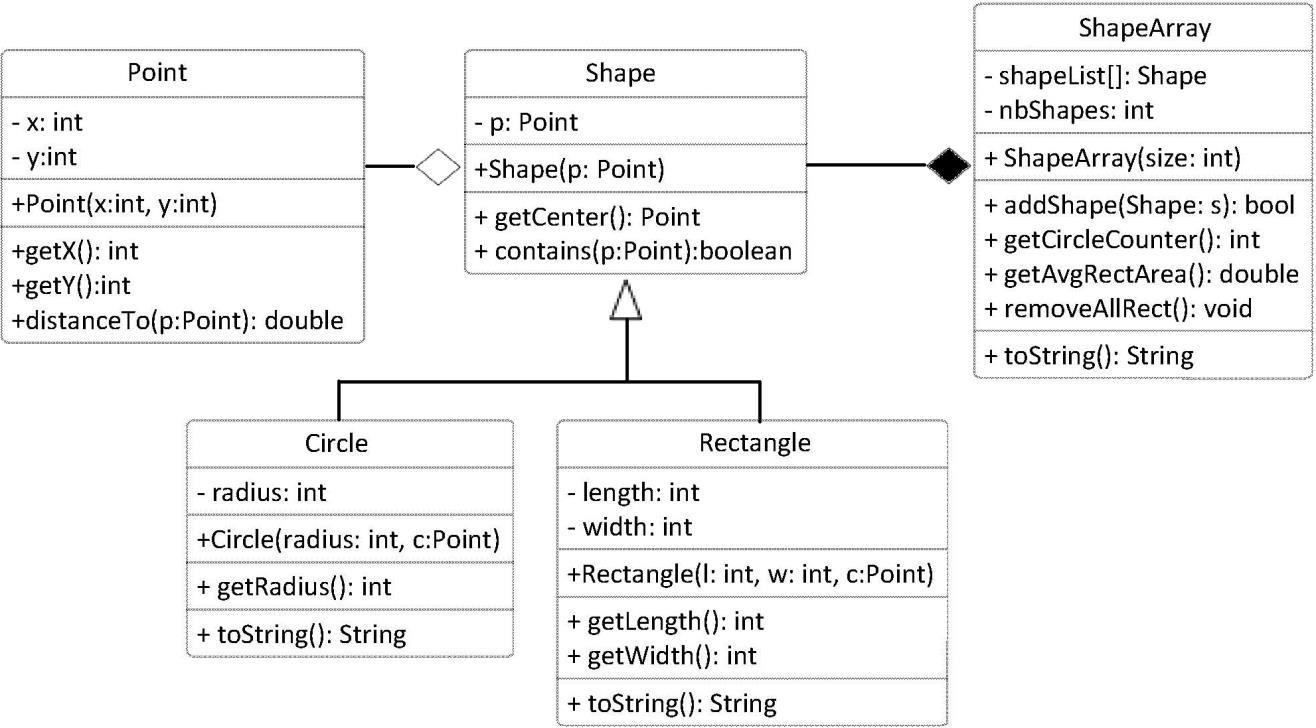
|  |
| --- |
| public class PayRollSystemTest { public static void main (String [] args) {    Employee firstEmployee = new SalariedEmployee("Usman" ,"Ali","11111-1111", 800.00 );    Employee secondEmployee = new CommissionEmployee("Atif" ,"Aslam", |

|  |
| --- |
| "222-22-2222", 10000, 0.06 );  Employee thirdEmployee = new BasePlusCommissionEmployee("Rana",  "Naseeb", "333-33-3333", 5000 , 0.04 , 300 );  Employee fourthEmployee = new HourlyEmployee( "Renson" , "Isaac",  "444-44-4444" , 16.75 , 40 );  // polymorphism: calling toString() and earning() on Employee’s reference  System.out.println(firstEmployee);  System.out.println(firstEmployee.earnings());  System.out.println(secondEmployee);  System.out.println(secondEmployee.earnings());  System.out.println(thirdEmployee);  // performing downcasting to access & raise base salary  BasePlusCommissionEmployee currentEmployee = (BasePlusCommissionEmployee) thirdEmployee;  double oldBaseSalary = currentEmployee.getBaseSalary(); System.out.println( "old base salary: " + oldBaseSalary) ;  currentEmployee.setBaseSalary(1.10 \* oldBaseSalary);  System.out.println("new base salary with 10% increase is:"+ currentEmployee.getBaseSalary());  System.out.println(thirdEmployee.earnings() );  System.out.println(fourthEmployee);  System.out.println(fourthEmployee.earnings() );  } // end main  } // end class |

**Output:**

****

**Exercise 2 (a)**



Implement classes: Shape, Circle and Rectangle based on the class diagram and description below:

Class Point implementation is given as follow:

|  |
| --- |
| class Point { private int x; private int y;  public Point(int x, int y) { this.x = x; this.y = y;  }  public int getX() { return x;} public int getY() { return y;} public double distanceTo(Point p) { return Math.sqrt((x-p.getX())\*(x-p.getX())+  (y-p.getY())\*(y-p.getY()));  }  public String toString() { return "("+x+", "+y+")";  }  } |

Class **Shape** has:

* An attributes of type Point, specifies the center of the shape object.
* A constructor that allows to initialize the center attribute with the value of the passed parameter
* A method that takes an object of type Point as a parameter and returns true if the point resides within the shape’s area, and false otherwise.

Class **Circle** has:

* An attribute of type integer specifies the radius measure of the circle

* A constructor that takes a Point parameter to initialize the center and an integer parameter to initialize the radius
* A getRadius method to return the value of the attribute radius
* An overriding version of toString method to return the attribute values of a Circle object as String

Class **Rectangle** has:

* Two integer attributes represents the length and width of the Rectangle object
* A constructor to initialize the center, length and width attribute for a new Rectangle object
* Methods getLength and getWidth returns the values of attributes length and width respectively
* An overriding version of toString method to return the attribute values of a Rectangle object as a String Class **ShapesArray**
* displayrectsinfo() →display all rectangles information
* getCirclecounter():int →return the number of circles
* getAvgAreas():double →return the average area of all shapes
* removeallrect() →delete all rectangles

**Source Code:**

**Class Shape:**

public class Shape {

private Point center;

public Shape(Point center) {

this.center = center;

}

public Point getCenter() {

return center;

}

public boolean contains(Point p) {

return false;

}

public double getArea() {

return 0.0;

}

}

**Class Circle:**

public class Circle extends Shape {

private int radius;

public Circle(Point center, int radius) {

super(center);

this.radius= radius;

}

@Override

public double getArea() {

return Math.PI \* radius \* radius;

}

@Override

public String toString() {

return "Circle: Center= " + getCenter() + ", radius=" + radius;

}

}

**Class Rectangle:**

public class Rectangle extends Shape{

private int length;

private int width;

public Rectangle(Point center, int length, int width) {

super(center);

this.length = length;

this.width = width;

}

public int getLength() {

return length;

}

public int getWidth() {

return width;

}

@Override

public double getArea() {

return length \* width;

}

@Override

public String toString() {

return "Rectangle: Center= " + getCenter() + ", length=" + length + ", width=" + width;

}

}

**Class Point:**

public class Point {

private int x;

private int y;

public Point(int x, int y) {

this.x = x;

this.y = y;

}

public int getX() {

return x;

}

public int getY() {

return y;

}

public double distanceTo(Point p) {

return Math.sqrt((x-p.getX())\*(x-p.getX())+

(y-p.getY())\*(y-p.getY()));

}

@Override

public String toString() {

return "("+x+", "+y+")";

}

}

**Class ShapesArray:**

public class ShapesArray {

private Shape[] shapeList;

private int numShapes;

public ShapesArray(int size) {

shapeList = new Shape[size];

numShapes = 0;

}

public boolean addShape(Shape s) {

if (numShapes < shapeList.length) {

shapeList[numShapes++] = s;

return true;

}

return false;

}

public void displayRectsInfo() {

for (int i = 0; i < numShapes; i++) {

if (shapeList[i] instanceof Rectangle) {

System.out.println(shapeList[i]);

}

}

}

public int getCircleCounter() {

int count = 0;

for (int i = 0; i < numShapes; i++) {

if (shapeList[i] instanceof Circle) {

count++;

}

}

return count;

}

public double getAvgAreas() {

if (numShapes == 0) return 0.0;

double totalArea = 0;

for (int i = 0; i < numShapes; i++) {

totalArea += shapeList[i].getArea();

}

return totalArea / numShapes;

}

public void removeAllRects() {

int index = 0;

for (int i = 0; i < numShapes; i++) {

if (!(shapeList[i] instanceof Rectangle)) {

shapeList[index++] = shapeList[i];

}

}

numShapes = index;

}

}

**Exercise 2 (b)**

**Step 6: Putting it all Together**

Implementation TestShape as given.

create ShapesArray object with size=20 display these options

1. add new shape
   1. for rectangle (ask for details)
   2. for circle (ask for details)
2. display all rectangles
3. display the average shapes area
4. display the number of circles
5. remove all rectangles
6. exit

**Source Code:**

**Class TestShape:**

import java.util.Scanner;

public class TestShape {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

ShapesArray shapesArray = new ShapesArray(20);

while (true) {

System.out.println("Choose an option:");

System.out.println("1. Add new shape");

System.out.println("2. Display all rectangles");

System.out.println("3. Display the average shapes area");

System.out.println("4. Display the number of circles");

System.out.println("5. Remove all rectangles");

System.out.println("6. Exit");

int choice = scanner.nextInt();

switch (choice) {

case 1:

System.out.println("1. Add Rectangle\n2. Add Circle");

int shapeChoice = scanner.nextInt();

System.out.println("Enter center x and y:");

int x = scanner.nextInt();

int y = scanner.nextInt();

Point center = new Point(x, y);

if (shapeChoice == 1) {

System.out.println("Enter length and width:");

int length = scanner.nextInt();

int width = scanner.nextInt();

shapesArray.addShape(new Rectangle(center, length, width));

}

else if (shapeChoice == 2) {

System.out.println("Enter radius:");

int radius = scanner.nextInt();

shapesArray.addShape(new Circle(center, radius));

}

break;

case 2:

shapesArray.displayRectsInfo();

break;

case 3:

System.out.println("Average Area: " + shapesArray.getAvgAreas());

break;

case 4:

System.out.println("Number of Circles: " + shapesArray.getCircleCounter());

break;

case 5:

shapesArray.removeAllRects();

System.out.println("All rectangles removed.");

break;

case 6:

System.out.println("Exiting...");

scanner.close();

return;

default:

System.out.println("Invalid option. Try again.");

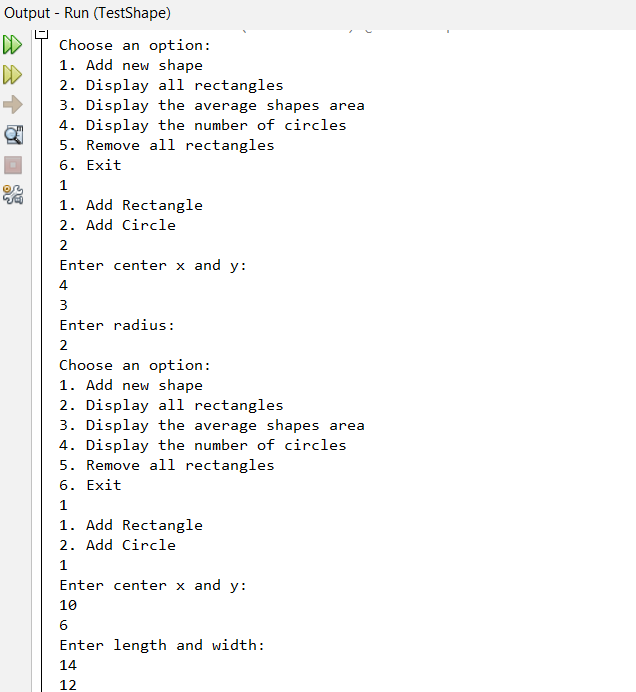
}

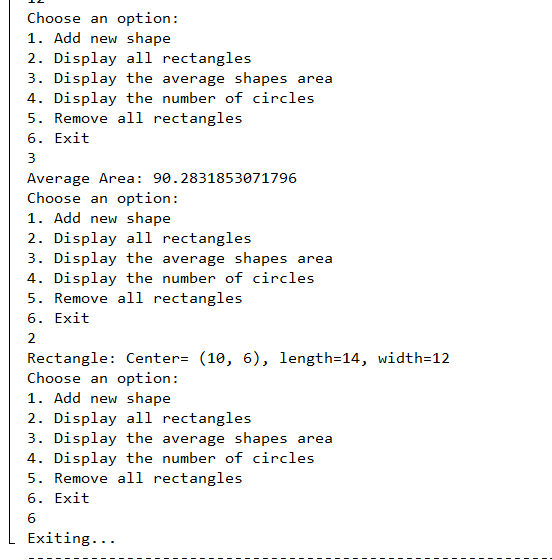
}

}

}

**Output:**

****

****