Lab 09 – Polymorphism

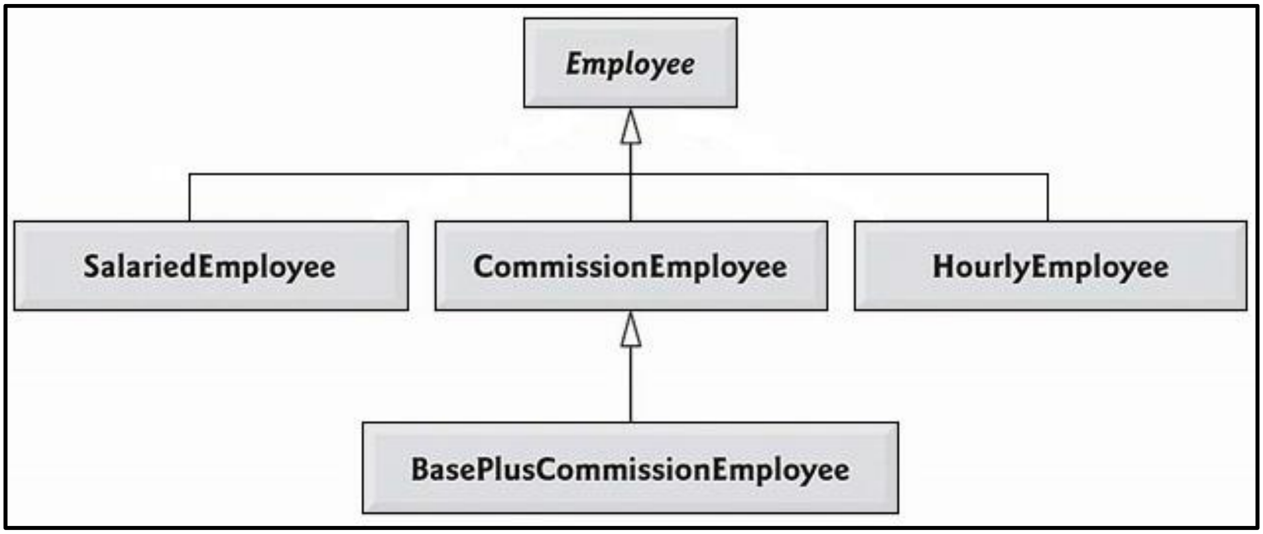
Task 01(a):

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 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;
       }
```

Code:

```
package Task01;
public class Employee {
    private String firstName;
    private String lastName;
    private String cnic;

    // default constructor
    public Employee(){}

    // parameterized constructor
    public Employee(String firstName, String lastName, String cnic){
        this.firstName = firstName;
        this.lastName = lastName;
        this.cnic = cnic;
    }

    // getters
    public String getFirstName(){
        return firstName;
    }
    public String getLastName(){
        return lastName;
    }
    public String getCnic() {
        return cnic;
    }
    // setters
    public void setFirstName(String firstName){
        this.firstName = firstName;
    }
    public void setLastName(String lastName){
        this.lastName = lastName;
    }
    public void setCnic(String cnic){
        this.cnic = cnic;
    }
    @Override
    public String toString(){
        return "Employee[" + firstName + ", " + lastName + ", " + cnic + "]";
    }
    public double earning(){
        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;
}
```

Code:

```
package Task01;

public class SalariedEmployee extends Employee{
    private double weeklySalary;

    // default constructor
    public SalariedEmployee(){}

    // parameterized constructor
    public SalariedEmployee(String firstName, String lastName, String cnic, 
double  weeklySalary){
        super(firstName, lastName, cnic);

        if(weeklySalary < 0){
            throw new IllegalArgumentException("Weekly salary cannot be 
negative.");
            // this will stop the program
        }
        this.weeklySalary = weeklySalary;
    }

    // getter
    public double getWeeklySalary(){
        return weeklySalary;
    }
```

```java
    // setter
    public void setWeeklySalary(double weeklySalary){
        if(weeklySalary < 0){
            throw new IllegalArgumentException("Weekly salary cannot be negative.");
        }
        this.weeklySalary = weeklySalary;
    }

    @Override
    public String toString(){
        return "SalariedEmplyee[" + weeklySalary + " " + super.toString() + "]";
    }

    @Override
    public double earning(){
        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{
         return 40*wage + (hours-40)*wage*1.5;
   }
}
```

Code:

```
package Task01;

public class HourlyEmployee extends Employee {
    private double hours;
    private double wage;

    // default constructor
    public HourlyEmployee() {
    }

    // parameterized constructor
    public HourlyEmployee(String firstName, String lastName, String cnic, double
hours, double wage) {
        super(firstName, lastName, cnic);

        if (hours < 0) {
            throw new IllegalArgumentException("Hours cannot be negative.");
        }
        this.hours = hours;

        if (wage < 0) {
            throw new IllegalArgumentException("Wage cannot be negative.");
        }
```

```java
        this.wage = wage;
    }

    // getters
    public double getHours() {
        return hours;
    }

    public double getWage() {
        return wage;
    }

    // setters
    public void setHours(double hours) {
        if (hours < 0) {
            throw new IllegalArgumentException("Hours cannot be negative.");
        }
        this.hours = hours;
    }

    public void setWage(double wage) {
        if (wage < 0) {
            throw new IllegalArgumentException("Wage cannot be negative.");
        }
        this.wage = wage;
    }

    @Override
    public String toString() {
        return "HourlyEmployee[" + hours + ", " + wage + " " + super.toString() +
"]";
    }

    @Override
    public double earning() {
        if (hours <= 40) {
            return wage * hours;
        } else {
            return 40 * wage + (hours - 40) * wage * 1.5;
        }
    }

}
```

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;
}
```

Code:

```
package Task01;

public class CommissionEmployee extends Employee {
    private double grossSales;
    private double commissionRate;

    // default constructor
    public CommissionEmployee() {
    }

    // parameterized constructor
    public CommissionEmployee(String firstName, String lastName, String cnic,
double grossSales, double commissionRate) {
        super(firstName, lastName, cnic);

        if (grossSales < 0) {
            throw new IllegalArgumentException("Gross sales cannot be negative");
        }
        this.grossSales = grossSales;

        if (commissionRate < 0) {
            throw new IllegalArgumentException("Commission rate cannot be
negative");
        }
        this.commissionRate = commissionRate;
    }

    // getters
    public double getGrossSales() {
        return grossSales;
    }

    public double getCommissionRate() {
        return commissionRate;
    }

    // setters
    public void setGrossSales(double grossSales) {
        if (grossSales < 0) {
            throw new IllegalArgumentException("Gross sales cannot be negative");
        }
        this.grossSales = grossSales;
    }

    public void setCommissionRate(double commissionRate) {
        if (commissionRate < 0) {
            throw new IllegalArgumentException("Commission rate cannot be
negative");
        }
        this.commissionRate = commissionRate;
    }

    @Override
    public String toString() {
        return "CommissionEmployee [" + grossSales + ", " + commissionRate + " "
+ super.toString() + "]";
    }

    @Override
    public double earning() {
        return grossSales * commissionRate;
    }

}
```

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();
}
```

Code:

```
package Task01;

public class BasePlusComissionEmployee extends CommissionEmployee{
    private double baseSalary;

    // default constructor
    public BasePlusComissionEmployee() { }

    // parameterized constructor
    public BasePlusComissionEmployee(String firstName, String lastName, String
cnic, double grossSales, double commissionRate, double baseSalary){
        super(firstName, lastName, cnic, grossSales, commissionRate);

        if(baseSalary < 0){
            throw new IllegalArgumentException("Base salary cannnot be
negative.");
        }
        this.baseSalary = baseSalary;
    }

    // getter
    public double getBaseSalary(){
        return baseSalary;
    }
    // setter
    public void setBaseSalary(double baseSalary){
        if(baseSalary < 0){
            throw new IllegalArgumentException("Base salary cannnot be 
negative.");
        }
        this.baseSalary = baseSalary;
    }

    @Override
    public String toString() {
        return "BasePlusCommissionEmployee [" + baseSalary + " " + 
super.toString() + "]";
    }

    @Override
    public double earning() {
        return baseSalary + super.earning();
    }
}
```

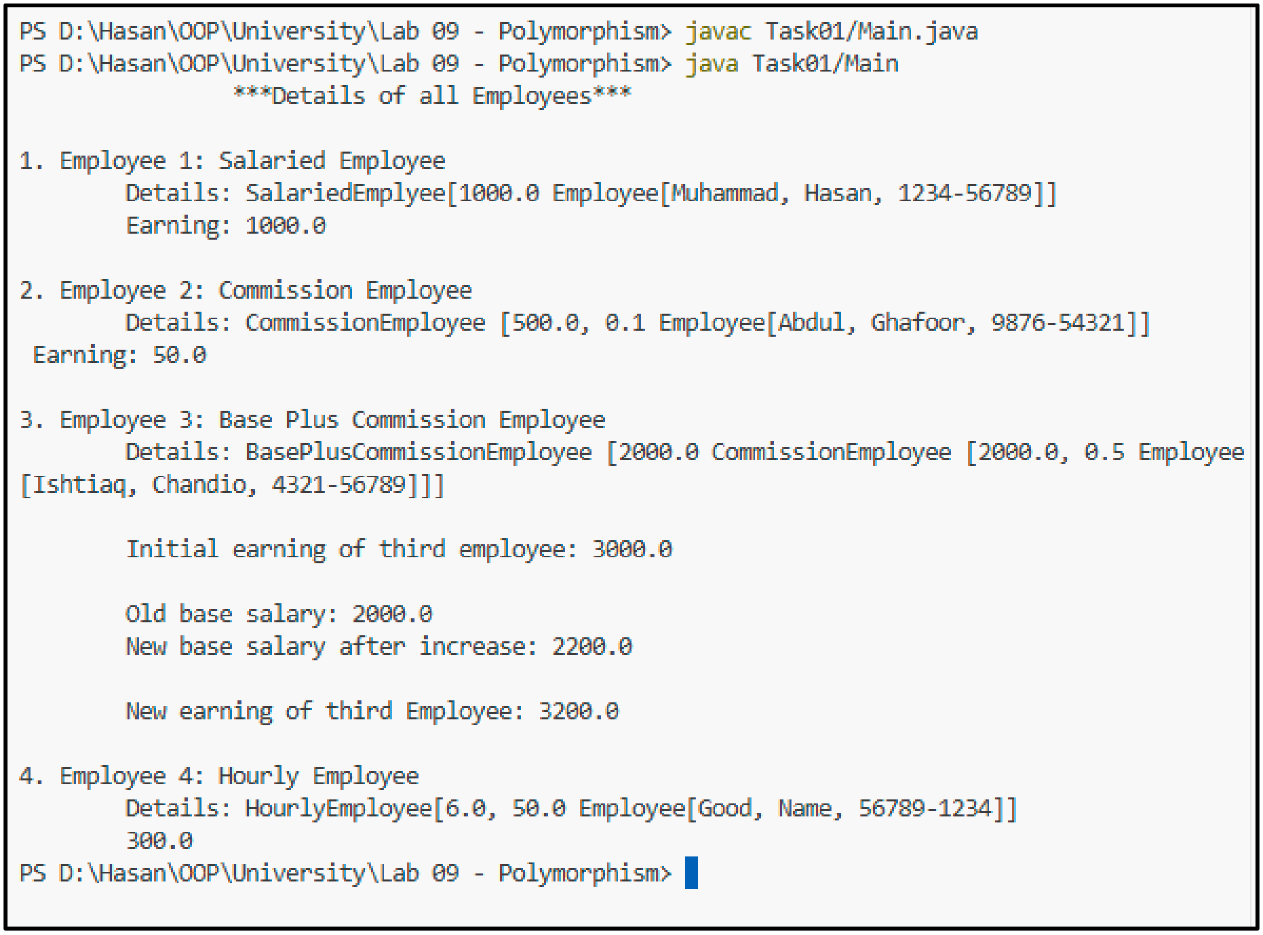
Task 01(b):

Step 6: Putting it all Together

Code:

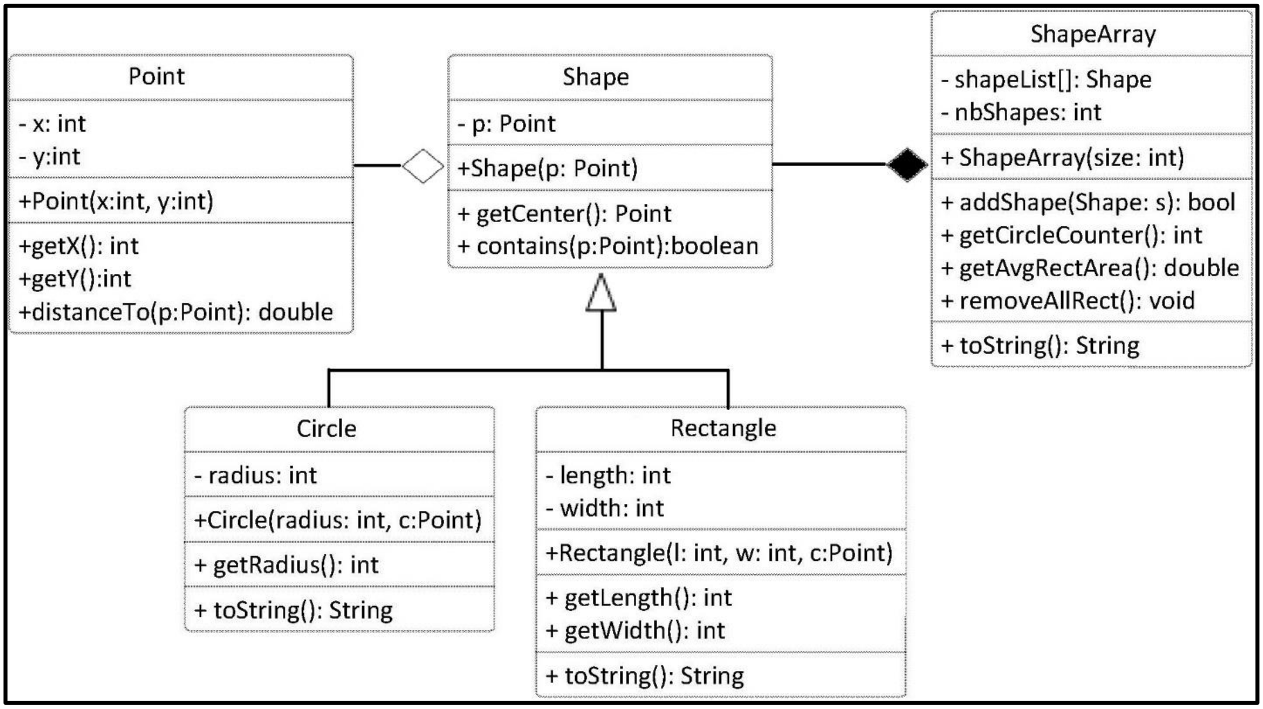
```
package Task01;

public class Main {
    public static void main(String[] args) {

        // creating one of each type
        Employee e1 = new SalariedEmployee("Muhammad", "Hasan", "1234-56789",
1000.0);
        Employee e2 = new CommissionEmployee("Abdul", "Ghafoor", "9876-54321",
500, 0.10);
        Employee e3 = new BasePlusComissionEmployee("Ishtiaq", "Chandio", "4321-
56789", 2000, 0.5, 2000);
        Employee e4 = new HourlyEmployee("Good", "Name", "56789-1234", 6, 50);

        System.out.println("\t\t***Details of all Employees***");

        /*
         * 1. displaying e1
         */

        System.out.println("\n1. Employee 1: Salaried Employee");
        System.out.println("\tDetails: " + e1);
        System.out.println("\t" + "Earning: " + e1.earning()); // it will call
the overridden earning() function

        /*
         * 2. displaying e2
         */
        System.out.println("\n2. Employee 2: Commission Employee");
        System.out.println("\tDetails: " + e2);
        System.out.println("\t" + "Earning: " + e2.earning());

        /*
         * 3. displaying e3
         */
        System.out.println("\n3. Employee 3: Base Plus Commission Employee");
        System.out.println("\tDetails: " + e3);
        System.out.println( "\n\t" + "Initial earning of third employee: " +
e3.earning());

        // performing downcasting to access and set base salary
        // initially we created e2 with the reference of Emplyee
        BasePlusComissionEmployee currEmp = (BasePlusComissionEmployee)e3;

        // first getting old base salary of our down casted employee
        double oldBaseSalary = currEmp.getBaseSalary();
        System.out.println("\n\t" + "Old base salary: " + oldBaseSalary);

        currEmp.setBaseSalary(oldBaseSalary*1.10);
        System.out.println("\t" + "New base salary after increase: " +
currEmp.getBaseSalary());

        // displaying earning
        System.out.println("\n\t" + "New earning of third Employee: " +
e3.earning());

        /*
         * 4. displaying e4
         */
        System.out.println("\n4. Employee 4: Hourly Employee");
        System.out.println("\tDetails: " + e4);
        System.out.println("\t" + e4.earning());
    }
}
```

Output:



```
PS D:\Hasan\OOP\University\Lab 09 - Polymorphism> javac Task01/Main.java
PS D:\Hasan\OOP\University\Lab 09 - Polymorphism> java Task01/Main
                ***Details of all Employees***

1. Employee 1: Salaried Employee
        Details: SalariedEmplyee[1000.0 Employee[Muhammad, Hasan, 1234-56789]]
        Earning: 1000.0

2. Employee 2: Commission Employee
        Details: CommissionEmployee [500.0, 0.1 Employee[Abdul, Ghafoor, 9876-54321]]
 Earning: 50.0

3. Employee 3: Base Plus Commission Employee
        Details: BasePlusCommissionEmployee [2000.0 CommissionEmployee [2000.0, 0.5 Employee
[Ishtiaq, Chandio, 4321-56789]]]

        Initial earning of third employee: 3000.0

        Old base salary: 2000.0
        New base salary after increase: 2200.0

        New earning of third Employee: 3200.0

4. Employee 4: Hourly Employee
        Details: HourlyEmployee[6.0, 50.0 Employee[Good, Name, 56789-1234]]
        300.0
PS D:\Hasan\OOP\University\Lab 09 - Polymorphism> 
```

Task 02(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+")";
      }
}
```

Code:

```
package Task02;

public class Point {
    private int x;
    private int y;

    // constructor
    public Point(int x, int y){
        this.x = x;
        this.y = y;
    }

    // getters
    public int getX(){
        return x;
    }
    public int getY(){
        return y;
    }

    @Override
    public String toString(){
        return "Point[" + x + ", " + y + "]";
    }

    public double distanceTo(Point p){
        return Math.sqrt( (x-p.getX())*(x-p.getX()) + (y-p.getY())*(y-p.getY())
);
    }
}
```

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

Code:

```
package Task02;

abstract public class Shape {
    protected Point p;

    // constructor
    public Shape(Point p){
        this.p = p;
    }

    // getter
    public Point getCenter(){
        return p;
    }

    @Override
    public String toString(){
        return "Shape [" + p + "]";
    }

    // this method will be implemented by sub classes
    abstract boolean contains(Point p);

}
```

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

Code:

```java
package Task02;

public class Circle extends Shape{
    private int radius;

    // constructor
    public Circle(Point p, int radius){
        super(p);
        this.radius = radius;
    }

    // getter
    public int getRadius(){
        return radius;
    }

    @Override
    public String toString() {
        return "Circle [" + radius + " " + super.toString() + "]";
    }

    @Override
    boolean contains(Point p) {
        // Point is inside circle if distance from center <= radius
        double distance = this.p.distanceTo(p);
        return distance <= radius;
    }
}
```

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

Code:

```
package Task02;

public class Rectangle extends Shape {
    private int length;
    private int width;

    // constructor
    public Rectangle(Point p, int length, int width) {
        super(p);
        this.length = length;
        this.width = width;
    }

    // getters
    public int getLength() {
        return length;
    }

    public int getWidth() {
        return width;
    }

    @Override
    public String toString() {
        return "Rectangle[" + length + ", " + width + " " + super.toString() + 
"]";
    }

    @Override
    boolean contains(Point p) {
        // Calculate bounds of rectangle based on center point
        int halfLength = length / 2;
        int halfWidth = width / 2;

        int left = this.p.getX() - halfWidth;
        int right = this.p.getX() + halfWidth;
        int top = this.p.getY() + halfLength;
        int bottom = this.p.getY() - halfLength;

        // Check if point is within bounds
        return p.getX() >= left && p.getX() <= right && p.getY() >= bottom &&
p.getY() <= top;
    }
}
```

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

Code:

```
package Task02;


public class ShapeArray {
    private Shape[] shapeList;
    private int noOfShapes;

    // constructor
    public ShapeArray(int size){
        shapeList = new Shape[size];
        noOfShapes = 0;
    }

    @Override
    public String toString() {
        return "ShapeList [ " + shapeList.toString() + ", " + noOfShapes + "]";
    }

    // 1. method to add shape
    public boolean addShape(Shape s){
        if(noOfShapes >= shapeList.length) return false;

        shapeList[noOfShapes] = s;
        noOfShapes++;
        return true;
    }
    // 2.  this method will return number of circles
    public int getCircleCounter(){
        int count = 0; // mandatory to intialize

        for(int i = 0; i < noOfShapes; i++){
            if(shapeList[i] instanceof Circle){
                count++;
            }
        }
        return count;
    }

    // 3. get average of all rectangle's area
    public double getAvgRectArea(){
        int rectangleCount = 0;
        double totalArea = 0.00;

        for(int i = 0; i < noOfShapes; i++){
            if(shapeList[i] instanceof Rectangle){
                Rectangle rect = (Rectangle)shapeList[i];

                totalArea += rect.getLength()*rect.getWidth();
                rectangleCount++;
            }
        }

        if(rectangleCount == 0) return 0.00; // what if we don't have any
rectangle in our shapeList

        return totalArea/rectangleCount;
    }

    // 4. method to remove all rectangles
    public void removeAllRect(){
        for(int i = 0; i < noOfShapes; i++){

            if(shapeList[i] instanceof Rectangle){
                for(int j = i; j < noOfShapes - 1; j++){
                    shapeList[j] = shapeList[j+1];
                }
            }
            noOfShapes--;
        }
    }
}
```

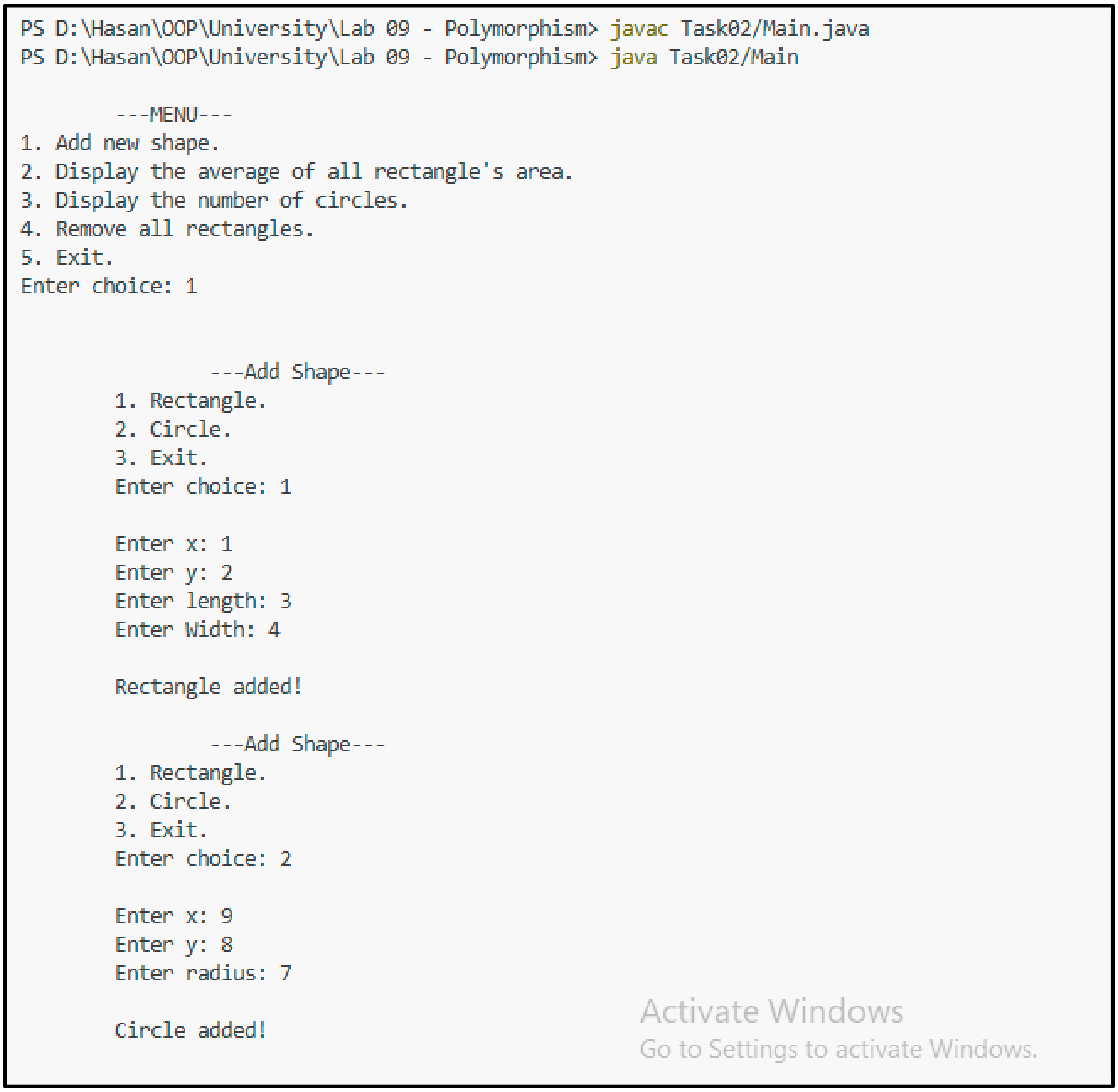
Task 02(b):

Putting it all Together

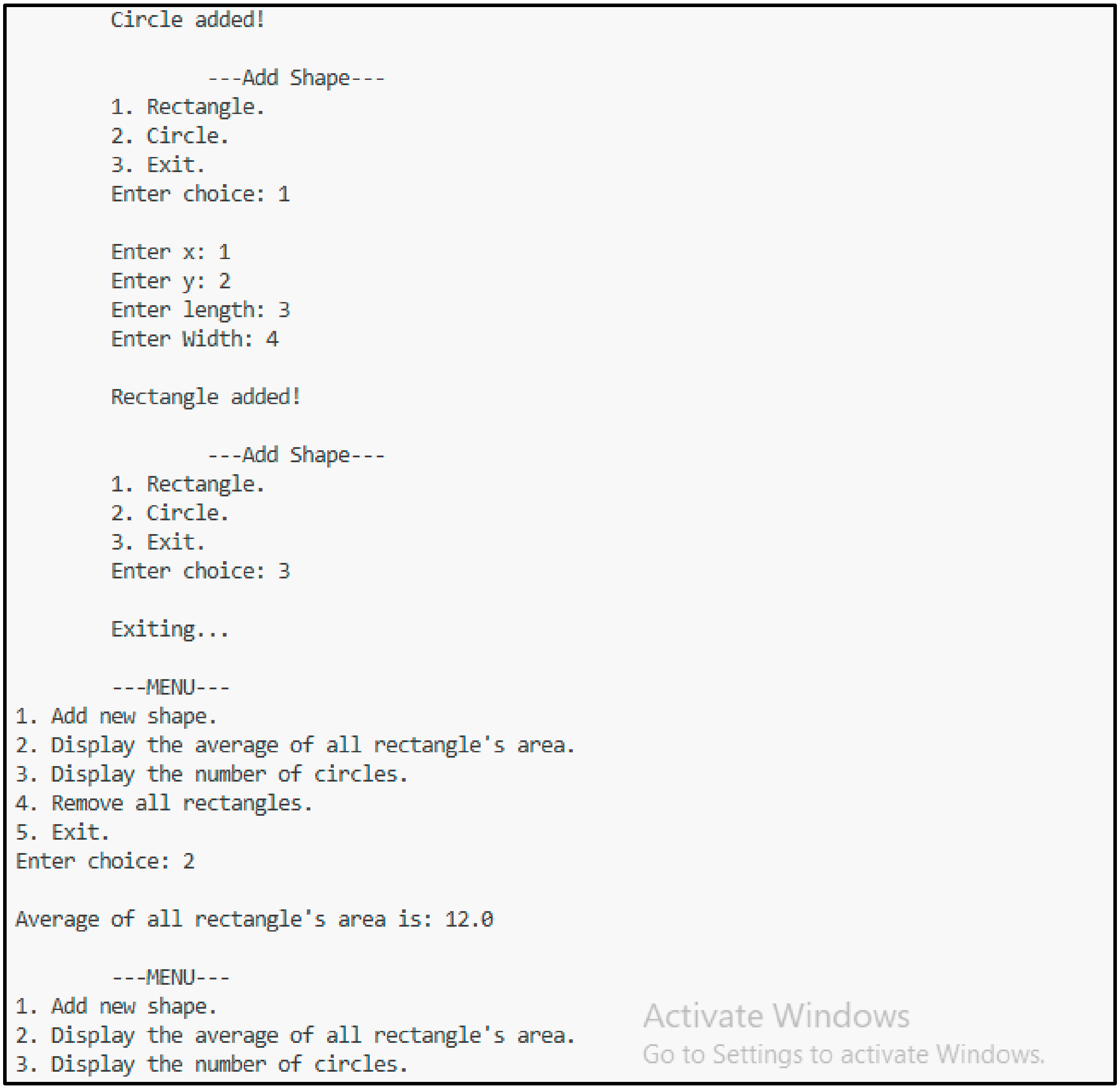
Code:

```
package Task02;
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        // create ShapeArray object with size = 20
        ShapeArray shapeArray = new ShapeArray(20);


        int choice = 0;

        do {
            System.out.println("\n\t---MENU---");
            System.out.println("1. Add new shape.");
            System.out.println("2. Display the average of all rectangle's
area.");
            System.out.println("3. Display the number of circles.");
            System.out.println("4. Remove all rectangles.");
            System.out.println("5. Exit.");
            System.out.print("Enter choice: ");
            choice = sc.nextInt();
            System.out.println();

            if(choice == 1){
                int choice1 = 0;

                do {
                    System.out.println("\n\t\t---Add Shape---");
                    System.out.println("\t1. Rectangle.");
                    System.out.println("\t2. Circle.");
                    System.out.println("\t3. Exit.");
                    System.out.print("\tEnter choice: ");
                    choice1 = sc.nextInt();
                    System.out.println();

                    // for rectangle
                    if(choice1 == 1){
                        System.out.print("\tEnter x: ");
                        int x = sc.nextInt();
                        System.out.print("\tEnter y: ");
                        int y = sc.nextInt();

                        // creating point
                        Point point = new Point(x, y);

                        System.out.print("\tEnter length: ");
                        int length = sc.nextInt();
                        System.out.print("\tEnter Width: ");
                        int width = sc.nextInt();

                        // creating rectangle
                        Rectangle rectangle = new Rectangle(point, length, 
width);

                        // adding it into shapeArray
                        if(shapeArray.addShape(rectangle)){
                            System.out.println("\n\tRectangle added!");
                        }
                        else System.out.println("\n\tRectangle not added.");

                    }

                    // for circle
                    else if(choice1 == 2){
                        System.out.print("\tEnter x: ");
                        int x = sc.nextInt();
                        System.out.print("\tEnter y: ");
                        int y = sc.nextInt();

                        // creating point
                        Point point = new Point(x, y);

                        System.out.print("\tEnter radius: ");
                        int radius = sc.nextInt();

                        Circle circle = new Circle(point, radius);
                        if(shapeArray.addShape(circle)){
                            System.out.println("\n\tCircle added!");
                        }
                        else System.out.println("\n\tCircle not added.");
                    }
                    // exit
                    else if(choice1 == 3) System.out.println("\tExiting...");
                    else System.out.println("\tInvalid choice.");

                } while (choice1 != 3);

            }

            // choice 2 -> average of all rectangles area
            else if(choice == 2){
                System.out.println("Average of all rectangle's area is: " +
shapeArray.getAvgRectArea());
            }

            // choice 3 ->  no of all circles
            else if(choice == 3){
                System.out.println("Total no of all circles is: " +
shapeArray.getCircleCounter());
            }

            // choice 4 -> remove all rectangles
            else if(choice == 4){
                shapeArray.removeAllRect();
                System.out.println("All rectangles removed.");
            }

            // choice 5 -> exit
            else if(choice == 5) System.out.println("Exiting program.");

            else System.out.println("Invalid choice.");

        } while (choice != 5);

        sc.close();
    }
}
```

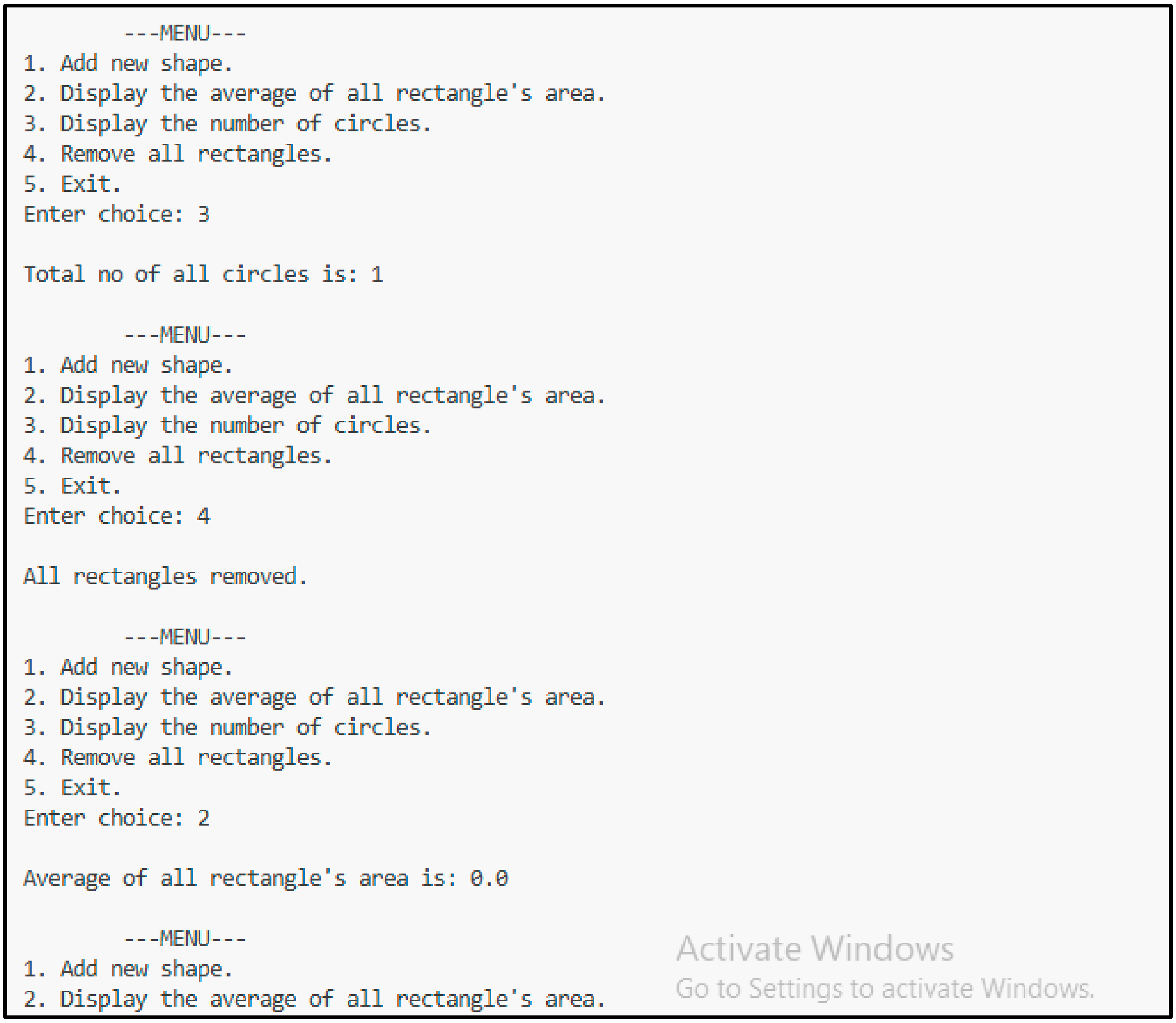
Output:



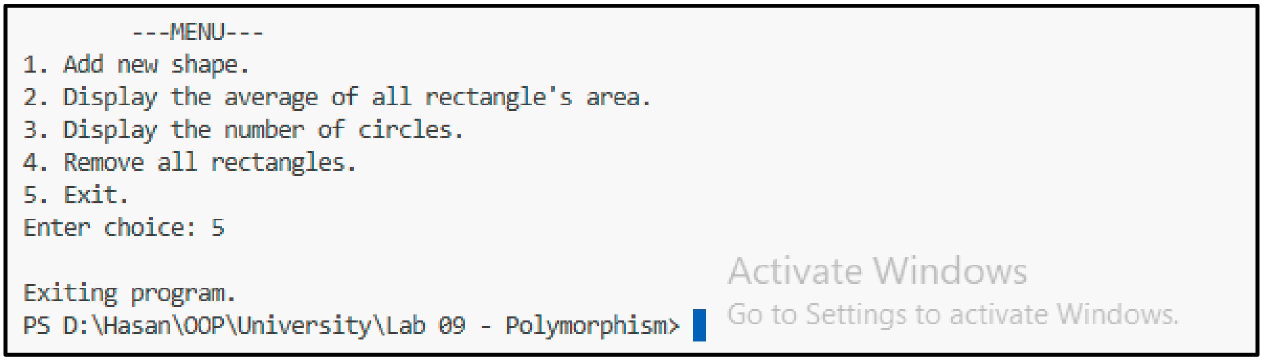
```
PS D:\Hasan\OOP\University\Lab 09 - Polymorphism> javac Task02/Main.java
PS D:\Hasan\OOP\University\Lab 09 - Polymorphism> java Task02/Main

        ---MENU---
1. Add new shape.
2. Display the average of all rectangle's area.
3. Display the number of circles.
4. Remove all rectangles.
5. Exit.
Enter choice: 1


                ---Add Shape---
        1. Rectangle.
        2. Circle.
        3. Exit.
        Enter choice: 1

        Enter x: 1
        Enter y: 2
        Enter length: 3
        Enter Width: 4

        Rectangle added!

                ---Add Shape---
        1. Rectangle.
        2. Circle.
        3. Exit.
        Enter choice: 2

        Enter x: 9
        Enter y: 8
        Enter radius: 7

        Circle added!
```



```
        Circle added!

                ---Add Shape---
        1. Rectangle.
        2. Circle.
        3. Exit.
        Enter choice: 1

        Enter x: 1
        Enter y: 2
        Enter length: 3
        Enter Width: 4

        Rectangle added!

                ---Add Shape---
        1. Rectangle.
        2. Circle.
        3. Exit.
        Enter choice: 3

        Exiting...

        ---MENU---
1. Add new shape.
2. Display the average of all rectangle's area.
3. Display the number of circles.
4. Remove all rectangles.
5. Exit.
Enter choice: 2

Average of all rectangle's area is: 12.0

        ---MENU---
1. Add new shape.
2. Display the average of all rectangle's area.
3. Display the number of circles.
```



```
        ---MENU---
1. Add new shape.
2. Display the average of all rectangle's area.
3. Display the number of circles.
4. Remove all rectangles.
5. Exit.
Enter choice: 3

Total no of all circles is: 1

        ---MENU---
1. Add new shape.
2. Display the average of all rectangle's area.
3. Display the number of circles.
4. Remove all rectangles.
5. Exit.
Enter choice: 4

All rectangles removed.

        ---MENU---
1. Add new shape.
2. Display the average of all rectangle's area.
3. Display the number of circles.
4. Remove all rectangles.
5. Exit.
Enter choice: 2

Average of all rectangle's area is: 0.0

        ---MENU---
1. Add new shape.
2. Display the average of all rectangle's area.
```



```
        ---MENU---
1. Add new shape.
2. Display the average of all rectangle's area.
3. Display the number of circles.
4. Remove all rectangles.
5. Exit.
Enter choice: 5

Exiting program.
PS D:\Hasan\OOP\University\Lab 09 - Polymorphism>
```