Problem 1
Code Explanation:

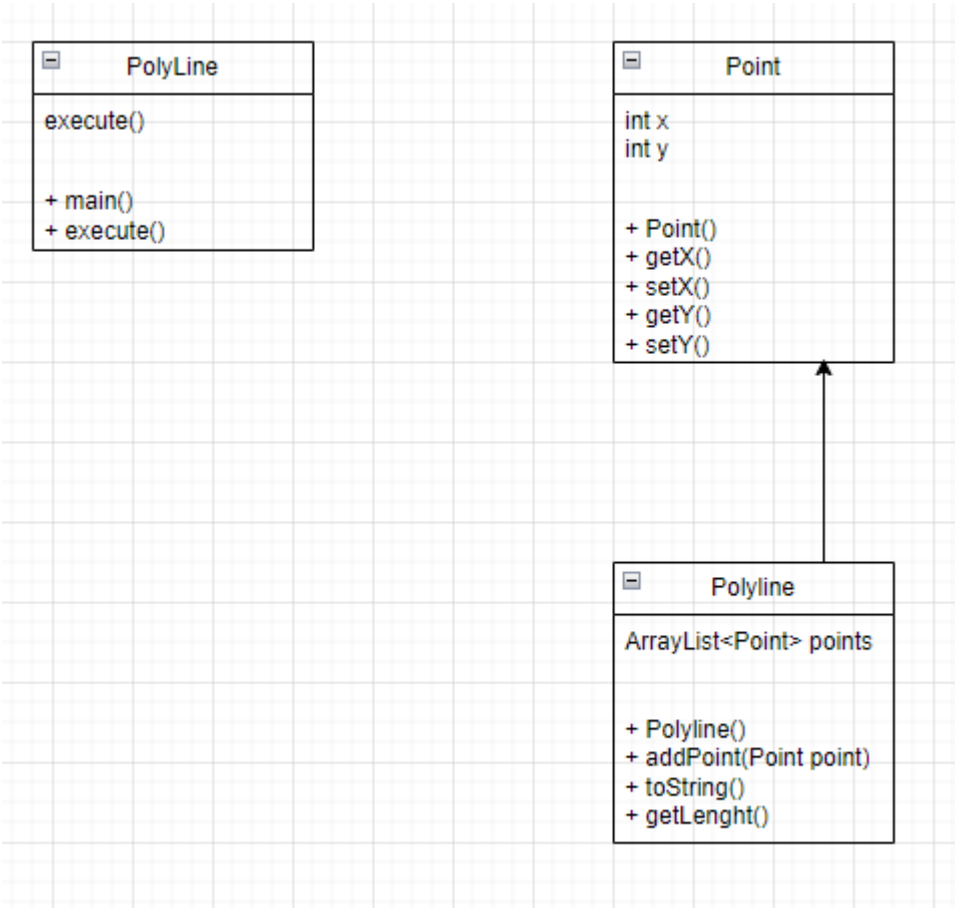
The PolyLine class contains the main method, where an instance of the PolyLine class is created, and it execute method is called.

The execute method creates an instance of the Polyline class and adds three points to it. Then, it prints the points in the polyline and calculates the length of the polyline.

The Point class represents a point with x and y coordinates. It has methods to get and set the x and y coordinates.

The Polyline class represents a sequence of connected points. It contains an ArrayList of Point objects.

The Polyline class has methods to add points, convert the polyline to a string representation, and calculate its length.



PolyLine contains the main method and an execute method.

Point represents a point with x and y coordinates.

Polyline represents a sequence of connected points. It has methods to add points, convert the polyline to a string representation, and calculate its length.

```java
import java.util.ArrayList;

public class PolyLine {
    public static void main(String[] args) {
        PolyLine main = new PolyLine(); // Create an instance of Main class
        main.execute(); // Call the execute method
    }

    public void execute() { // Instance method to avoid static context
        Polyline polyline = new Polyline();
        polyline.addPoint(new Point(0, 0));
        polyline.addPoint(new Point(1, 1));
        polyline.addPoint(new Point(2, 2));

        System.out.println("Points in polyline: " + polyline);
        polyline.getLength();
    }

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

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

        public int getX() {
            return this.x;
        }

        public void setX(int new_x) {
            this.x = new_x;
        }

        public int getY() {
            return this.y;
        }

        public void setY(int new_y) {
            this.y = new_y;
        }
    }

    static class Polyline {
        private ArrayList<Point> points;

        public Polyline() {
            this.points = new ArrayList<>();
        }

        public void addPoint(Point point) {
            points.add(point);
        }

        public String toString() {
            StringBuilder string = new StringBuilder();
            for (int i = 0; i < points.size(); i++) {
                string.append("(").append(points.get(i).getX()).append(",").append(points.get(i).getY()).append(")");
            }
            return string.toString();
        }

        public void getLength() {
            float total_length = 0;
            for (int i = 0; i < points.size() - 1; i++) {
                Point p1 = points.get(i);
                Point p2 = points.get(i + 1);
                total_length += Math.sqrt(Math.pow(p2.getX() - p1.getX(), 2) + Math.pow(p2.getY() - p1.getY(), 2));
            }
            System.out.println("Length of the polyline: " + total_length);
        }
    }
}
```

```
PS C:\Users\Kelvin\OneDrive\College\2n
cabb\bin' 'PolyLine'
Points in polyline: (0,0)(1,1)(2,2)
Length of the polyline: 2.828427
```

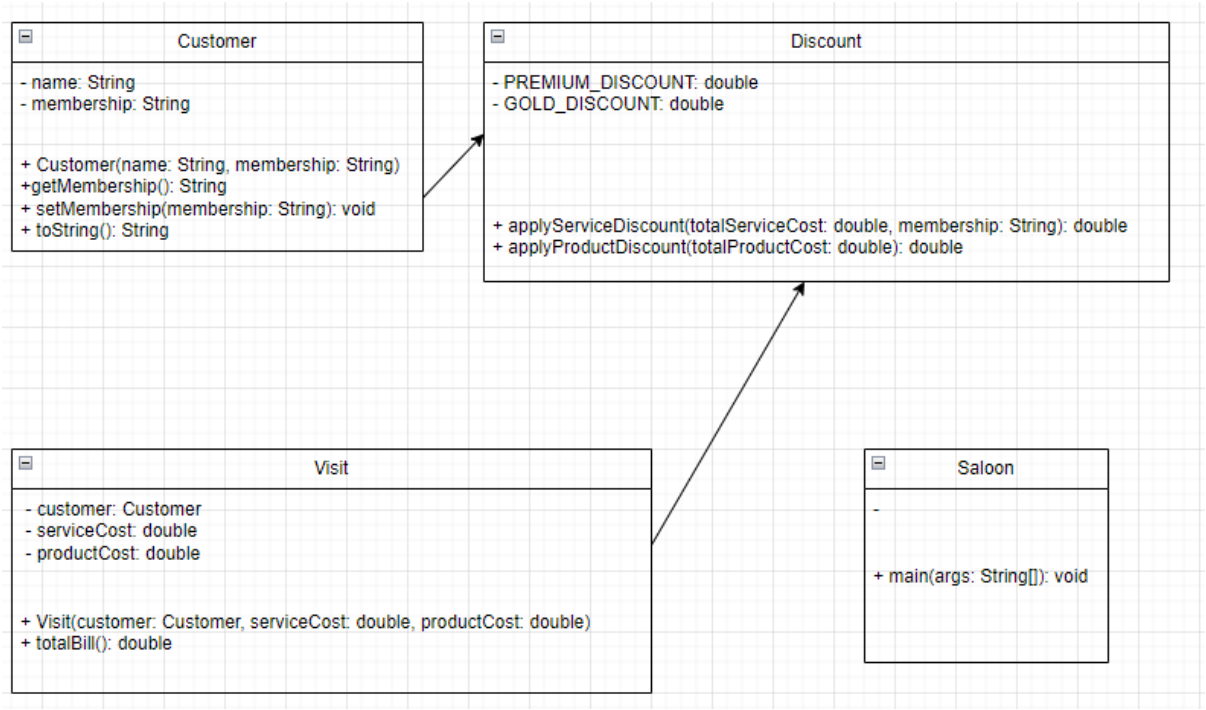
Problem 2
Code Explanation

The Customer class represents a customer with a name and a membership type.

The Discount class contains static methods to apply discounts on service and product costs based on the customer's membership.

The Visit class represents a visit to the salon by a customer, including the service and product costs. It calculates the total bill by applying discounts.

The Saloon class contains the main method to create customers, visits, and display the total bill for each visit.



In the diagram:

Customer Relation:

The Visit class has a composition relationship with the Customer class. This means that a Visit object is composed of a Customer object, as indicated by the customer: Customer attribute in the Visit class.

Visit Relation:

The Visit class is an aggregation of the Customer class. This means that a Visit object contains one or more Customer objects.

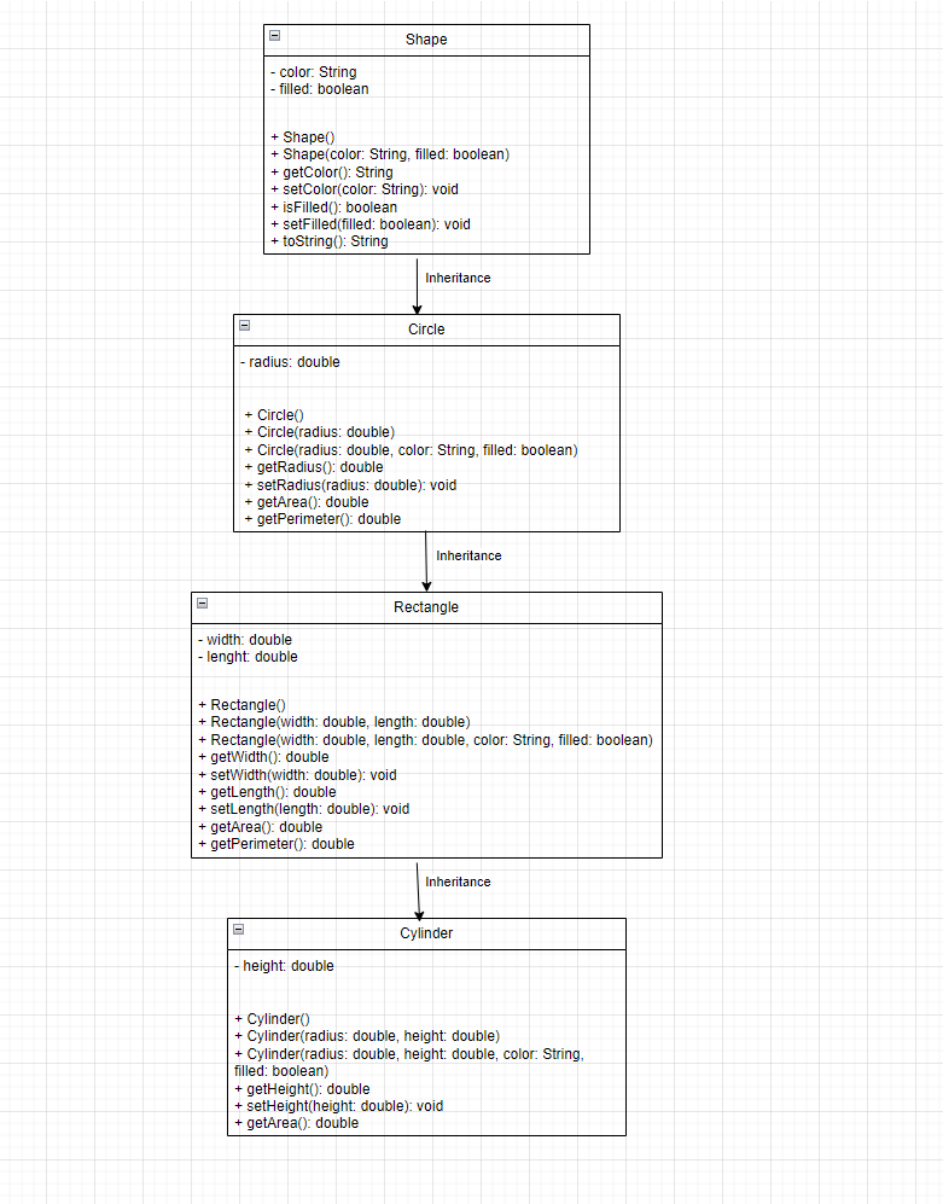
Saloon Relation:

The main method in the Saloon class has an association with the other classes (Customer, Visit, and possibly Discount). This means that the Saloon class interacts with instances of the other classes through method calls and object creations.

```java
class Customer {
    private String name;
    private String membership;

    public Customer(String name, String membership) {
        this.name = name;
        this.membership = membership;
    }

    public String getMembership() {
        return membership;
    }

    public void setMembership(String membership) {
        this.membership = membership;
    }

    @Override
    public String toString() {
        return "Customer: " + name + ", Membership: " + membership;
    }
}

class Discount {
    public static final double PREMIUM_DISCOUNT = 0.20;
    public static final double GOLD_DISCOUNT = 0.15;
    public static final double SILVER_DISCOUNT = 0.10;
    public static final double PRODUCT_DISCOUNT = 0.10;

    public static double applyServiceDiscount(double totalServiceCost, String membership) {
        if (membership != null) {
            switch (membership) {
                case "Premium":
                    return totalServiceCost * (1 - PREMIUM_DISCOUNT);
                case "Gold":
                    return totalServiceCost * (1 - GOLD_DISCOUNT);
                case "Silver":
                    return totalServiceCost * (1 - SILVER_DISCOUNT);
                default:
                    return totalServiceCost;
            }
        } else {
            // No membership, return original cost
            return totalServiceCost;
        }
    }

    public static double applyProductDiscount(double totalProductCost) {
        return totalProductCost * (1 - PRODUCT_DISCOUNT);
    }
}

class Visit {
    private Customer customer;
    private double serviceCost;
    private double productCost;

    public Visit(Customer customer, double serviceCost, double productCost) {
        this.customer = customer;
        this.serviceCost = serviceCost;
        this.productCost = productCost;
    }

    public double totalBill() {
        double totalServiceCost = Discount.applyServiceDiscount(serviceCost, customer.getMembership());
        double totalProductCost = Discount.applyProductDiscount(productCost);
        return totalServiceCost + totalProductCost;
    }
}

public class Saloon {
    public static void main(String[] args) {
        // Create customers
        Customer customer1 = new Customer("Alice", "Premium");
        Customer customer2 = new Customer("Bob", "Gold");
        Customer customer3 = new Customer("Charlie", "Silver");
        Customer customer4 = new Customer("David", null);

        // Create visits
        Visit visit1 = new Visit(customer1, 100, 50);
        Visit visit2 = new Visit(customer2, 100, 50);
        Visit visit3 = new Visit(customer3, 100, 50);
        Visit visit4 = new Visit(customer4, 100, 50);

        // Display bills
        System.out.println("Visit 1 Total Bill: " + visit1.totalBill());
        System.out.println("Visit 2 Total Bill: " + visit2.totalBill());
        System.out.println("Visit 3 Total Bill: " + visit3.totalBill());
        System.out.println("Visit 4 Total Bill: " + visit4.totalBill());
    }
}
```

```
Visit 1 Total Bill: 125.0
Visit 2 Total Bill: 130.0
Visit 3 Total Bill: 135.0
Visit 4 Total Bill: 145.0
```

Problem 3
Code Explanation



The Shape class serves as a base class for other shapes and contains attributes like colour and filled, along with methods to manipulate these attributes.

The Circle and Rectangle classes inherit from Shape and add specific attributes (radius for circles and width and length for rectangles) and methods to calculate their respective areas and perimeters.

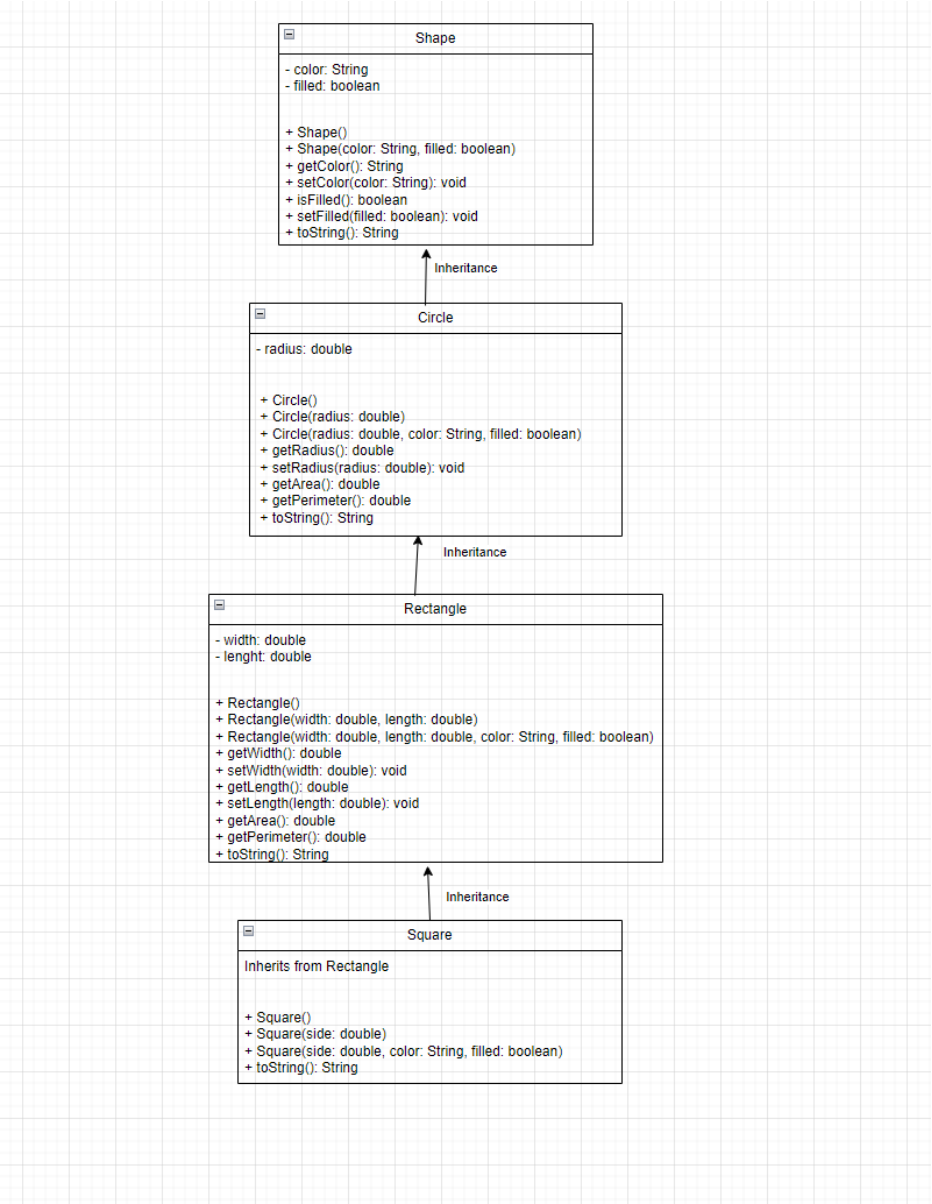
The Cylinder class inherits from Circle and adds the height attribute to represent the third dimension of the cylinder. It also overrides the getArea() method to calculate the surface area of the cylinder.

In the Shapes class, instances of each shape are created, and their properties are displayed, demonstrating the polymorphic behaviour of inheritance.

```java
class Shape {
    private String color;
    private boolean filled;

    public Shape() {
        this.color = "green";
        this.filled = true;
    }

    public Shape(String color, boolean filled) {
        this.color = color;
        this.filled = filled;
    }

    public String getColor() {
        return color;
    }

    public void setColor(String color) {
        this.color = color;
    }

    public boolean isFilled() {
        return filled;
    }

    public void setFilled(boolean filled) {
        this.filled = filled;
    }

    @Override
    public String toString() {
        return "A Shape with color of " + color + " and " + (filled ? "filled" : "Not filled");
    }
}

class Circle extends Shape {
    private double radius;

    public Circle() {
        this.radius = 1.0;
    }

    public Circle(double radius) {
        this.radius = radius;
    }

    public Circle(double radius, String color, boolean filled) {
        super(color, filled);
        this.radius = radius;
    }

    public double getRadius() {
        return radius;
    }

    public void setRadius(double radius) {
        this.radius = radius;
    }

    public double getArea() {
        return Math.PI * radius * radius;
    }

    public double getPerimeter() {
        return 2 * Math.PI * radius;
    }
}

class Rectangle extends Shape {
    private double width;
    private double length;

    public Rectangle() {
        this.width = 1.0;
        this.length = 1.0;
    }

    public Rectangle(double width, double length) {
        this.width = width;
        this.length = length;
    }

    public Rectangle(double width, double length, String color, boolean filled) {
        super(color, filled);
        this.width = width;
        this.length = length;
    }

    public double getWidth() {
        return width;
    }

    public void setWidth(double width) {
        this.width = width;
    }

    public double getLength() {
        return length;
    }

    public void setLength(double length) {
        this.length = length;
    }

    public double getArea() {
        return width * length;
    }

    public double getPerimeter() {
        return 2 * (width + length);
    }
}

class Cylinder extends Circle {
    private double height;

    public Cylinder() {
        super();
        this.height = 1.0;
    }

    public Cylinder(double radius, double height) {
        super(radius);
        this.height = height;
    }

    public Cylinder(double radius, double height, String color, boolean filled) {
        super(radius, color, filled);
        this.height = height;
    }

    public double getHeight() {
        return height;
    }

    public void setHeight(double height) {
        this.height = height;
    }

    @Override
    public double getArea() {
        return 2 * Math.PI * getRadius() * height + 2 * super.getArea();
    }
}

public class Shapes {
    public static void main(String[] args) {
        Shape shape1 = new Shape();
        System.out.println(shape1);

        Circle circle1 = new Circle(2.0, "blue", false);
        System.out.println(circle1);
        System.out.println("Area of circle: " + circle1.getArea());
        System.out.println("Perimeter of circle: " + circle1.getPerimeter());

        Rectangle rectangle1 = new Rectangle(2.0, 3.0, "red", true);
        System.out.println(rectangle1);
        System.out.println("Area of rectangle: " + rectangle1.getArea());
        System.out.println("Perimeter of rectangle: " + rectangle1.getPerimeter());

        Cylinder cylinder1 = new Cylinder(2.0, 3.0, "green", true);
        System.out.println(cylinder1);
        System.out.println("Area of cylinder: " + cylinder1.getArea());
    }
}
```

```
A Shape with color of green and filled
A Shape with color of blue and Not filled
Area of circle: 12.566370614359172
Perimeter of circle: 12.566370614359172
A Shape with color of red and filled
Area of rectangle: 6.0
Perimeter of rectangle: 10.0
A Shape with color of green and filled
Area of cylinder: 62.8318530717958 6
```

Problem 4
Code Explanation



The Shape class represents a basic shape with attributes colour and filled. It provides methods to access and modify these attributes.

Both Circle and Rectangle classes inherit from Shape and add specific attributes and methods for circles and rectangles, such as radius for circles and width and length for rectangles.

The Square class is a subclass of Rectangle as a square is a special case of a rectangle with equal width and length. It inherits all properties and methods from Rectangle and provides additional functionality to handle squares specifically.

In the Shapes2 class, instances of shapes are created, and their properties are displayed, demonstrating the use of inheritance and polymorphism.

```java
class Shape {
    private String color;
    private boolean filled;

    public Shape() {
        this.color = "green";
        this.filled = true;
    }

    public Shape(String color, boolean filled) {
        this.color = color;
        this.filled = filled;
    }

    public String getColor() {
        return color;
    }

    public void setColor(String color) {
        this.color = color;
    }

    public boolean isFilled() {
        return filled;
    }

    public void setfilled(boolean filled) {
        this.filled = filled;
    }

    @Override
    public String toString() {
        return "A Shape with color of " + color + " and " + (filled ? "filled" : "Not filled");
    }
}

class Circle extends Shape {
    private double radius;

    public Circle() {
        this.radius = 1.0;
    }

    public Circle(double radius) {
        this.radius = radius;
    }

    public Circle(double radius, String color, boolean filled) {
        super(color, filled);
        this.radius = radius;
    }

    public double getRadius() {
        return radius;
    }

    public void setRadius(double radius) {
        this.radius = radius;
    }

    public double getArea() {
        return Math.PI * radius * radius;
    }

    public double getPerimeter() {
        return 2 * Math.PI * radius;
    }

    @Override
    public String toString() {
        return "A Circle with radius=" + radius + ", which is a subclass of " + super.toString();
    }
}

class Rectangle extends Shape {
    private double width;
    private double length;

    public Rectangle() {
        this.width = 1.0;
        this.length = 1.0;
    }

    public Rectangle(double width, double length) {
        this.width = width;
        this.length = length;
    }

    public Rectangle(double width, double length, String color, boolean filled) {
        super(color, filled);
        this.width = width;
        this.length = length;
    }

    public double getWidth() {
        return width;
    }

    public void setWidth(double width) {
        this.width = width;
    }

    public double getLength() {
        return length;
    }

    public void setLength(double length) {
        this.length = length;
    }

    public double getArea() {
        return width * length;
    }

    public double getPerimeter() {
        return 2 * (width + length);
    }

    @Override
    public String toString() {
        return "A Rectangle with width=" + width + " and length=" + length + ", which is a subclass of " + super.toString();
    }
}

class Square extends Rectangle {

    public Square() {
        super();
    }

    public Square(double side) {
        super(side, side);
    }

    public Square(double side, String color, boolean filled) {
        super(side, side, color, filled);
    }

    @Override
    public void setWidth(double width) {
        super.setWidth(width);
        super.setLength(width);
    }

    @Override
    public void setLength(double length) {
        super.setWidth(length);
        super.setLength(length);
    }

    @Override
    public String toString() {
        return "A Square with side=" + getWidth() + ", which is a subclass of " + super.toString();
    }

    // No need to override getArea() and getPerimeter() since they work as expected for a square
}

public class Shapes2 {
    public static void main(String[] args) {
        Shape shape1 = new Shape();
        System.out.println(shape1);

        Circle circle1 = new Circle(2.0, "blue", false);
        System.out.println(circle1);
        System.out.println("Area of circle: " + circle1.getArea());
        System.out.println("Perimeter of circle: " + circle1.getPerimeter());

        Rectangle rectangle1 = new Rectangle(2.0, 3.0, "red", true);
        System.out.println(rectangle1);
        System.out.println("Area of rectangle: " + rectangle1.getArea());
        System.out.println("Perimeter of rectangle: " + rectangle1.getPerimeter());

        Square square1 = new Square(4.0, "yellow", true);
        System.out.println(square1);
        System.out.println("Area of square: " + square1.getArea());
        System.out.println("Perimeter of square: " + square1.getPerimeter());
    }
}
```

```
A Shape with color of green and filled
A Shape with color of blue and Not filled
Area of circle: 12.566370614359172
Perimeter of circle: 12.566370614359172
A Shape with color of red and filled
Area of rectangle: 6.0
Perimeter of rectangle: 10.0
A Square with side=4.0, which is a subclass of A Shape with color of yellow and filled
Area of square: 16.0
Perimeter of square: 16.0
```