SHEET 5 SOLUTIONS

SHEHAB MAHMOUD SALAH | 2100320

GENERAL NOTE: to **copy** code **from** this PDF document, **copy each block of code** separately to **not** lose the code's formatting, alternatively you can find all the **source code** to **PROGRAMMING EXERCISES** on my **GitHub**: https://bit.ly/CSE231Sheets , **happy compiling!**

1. What are the mistakes in the following program? Suggest a way to fix all errors and show the exact output after fixing it.

```java
class A {
    public void f1() {
        System.out.println("A.f1 called");
    }
    public final void f2() {
        System.out.println("A.f2 called");
    }
    public abstract void f3();
}

class B extends A {
    public void f1() {
        super.f1();
        System.out.println("B.f1 called");
    }
    public void f2() {
        super.f2();
        System.out.println("B.f2 called");
    }
    public void f3() {
        System.out.println("B.f3 called");
    }
}

class C extends A {
    public void f1() {
        super.f1();
        System.out.println("C.f1 called");
    }
}

abstract class D extends C {
    public void f3() {
        System.out.println("D.f3 called");
    }
}

final class E extends D {
}

class F extends E {
    public void f3() {
        super.f3();
        System.out.println("F.f3 called");
    }
}

public class Test {
    public static void main(String[] args) {
        A[] ps = new A[6];
        ps[0] = new A();
        ps[1] = new B();
        ps[2] = new C();
        ps[3] = new D();
        ps[4] = new E();
        ps[5] = new F();
        for (int i = 0; i < ps.length; i++) {
            ps[i].f1();
            ps[i].f2();
            ps[i].f3();
        }
    }
}
```

Abstract method in non-abstract class is not allowed.

Possible fix: We can remove the "abstract" keyword from the void method "f3" and leave it empty.

f2 is a final method and can't be overridden!

Possible fix: We could remove the "final" keyword from the f2 method.

Class E is a final class and can't be extended!

Possible fix: We could remove the "final" keyword from class E to extend it.

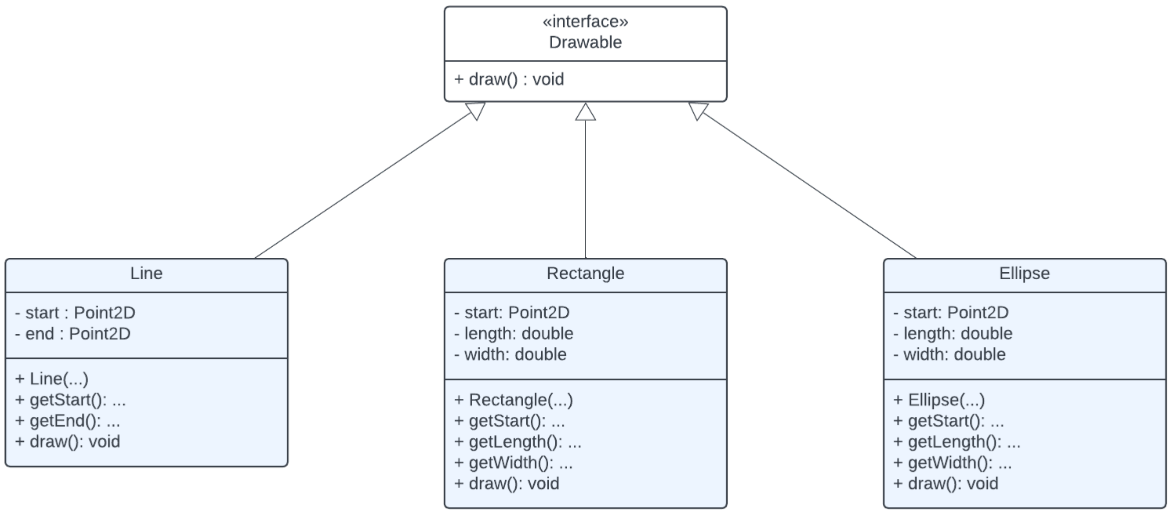
Class D is abstract and can't be instantiated!

Possible fix: Remove the D class instantiation.
NOTE: Removal of the D object instantiation would mean effectively lowering the bounds for the "ps" array, meaning we'd instantiate A[5] instead of A[6] and adjusting the indices of each reference element declared as well.

-> After applying the modifications mentioned above, the output would be:

```
A.f1 called
A.f2 called
A.f1 called
B.f1 called
A.f2 called
B.f2 called
B.f3 called
A.f1 called
C.f1 called
A.f2 called
A.f1 called
C.f1 called
A.f2 called
D.f3 called
A.f1 called
C.f1 called
A.f2 called
D.f3 called
F.f3 called
```

2. A possible UML for the code's hierarchy would be:



Implementing in code:

(**Drawable.java** on GitHub)

```
public interface Drawable{
    void draw();
}
```

(**Line.java** on GitHub)

```
import javafx.geometry.Point2D;

public class Line implements Drawable {
    private Point2D start;
    private Point2D end;

    public Line(Point2D start, Point2D end) {
        this.start = start;
```

```java
        this.end = end;
    }

    public Point2D getStart() {
        return start;
    }

    public Point2D getEnd() {
        return end;
    }

    @Override
    public void draw() {
        System.out.println("Drawing Line from " + start + " to " + end);
    }
}
```

(**Rectangle.java** on GitHub)

```java
import javafx.geometry.Point2D;

public class Rectangle implements Drawable {
    private Point2D start;
    private double length;
    private double width;

    public Rectangle(Point2D start, double length, double width) {
        this.start = start;
        this.length = length;
        this.width = width;
    }

    public Point2D getStart() {
        return start;
    }

    public double getLength() {
        return length;
    }

    public double getWidth() {
        return width;
    }

    @Override
    public void draw() {
        System.out.println("Drawing Rectangle at " + start + " with
length " + length + " and width " + width);
    }
}
```

(**Ellipse.java** on GitHub)

```java
import javafx.geometry.Point2D;

public class Ellipse implements Drawable {
    private Point2D start;
    private double length;
    private double width;

    public Ellipse(Point2D start, double length, double width) {
        this.start = start;
        this.length = length;
        this.width = width;
    }

    public Point2D getStart() {
        return start;
    }

    public double getLength() {
        return length;
    }

    public double getWidth() {
        return width;
    }

    @Override
    public void draw() {
        System.out.println("Drawing Ellipse at " + start + " with length
" + length + " and width " + width);
    }
```

3. We will define a new interface, “Shape” which will be implemented by *modified* versions of our previously declared Rectangle, Line and Ellipse classes, as follows:

(**Shape.java** on GitHub)

```java
import javafx.geometry.Point2D;
public interface Shape {
    void draw();
    Point2D getStartPoint();
}
```

(**LineModified.java** on GitHub)

```java
import javafx.geometry.Point2D;
public class LineModified implements Shape {
    private Point2D start;
    private Point2D end;

    public LineModified(Point2D start, Point2D end) {
        this.start = start;
        this.end = end;
    }

    @Override
    public Point2D getStartPoint() {
        return start;
    }
    @Override
    public void draw() {
        System.out.println("Drawing Line from " + start + " to " +
end);
    }
}
```

(**RectangleModified.java** on GitHub)

```
import javafx.geometry.Point2D;

public class RectangleModified implements Shape {
    private Point2D start;
    private double length;
    private double width;

    public RectangleModified(Point2D start, double length, double
width) {
        this.start = start;
        this.length = length;
        this.width = width;
    }

    @Override
    public Point2D getStartPoint() {
        return start;
    }

    @Override
    public void draw() {
        System.out.println("Drawing Rectangle at " + start + " with
length " + length + " and width " + width);
    }
}
```

(**EllipseModified.java** on GitHub)

```
import javafx.geometry.Point2D;

public class EllipseModified implements Shape {
    private Point2D start;
    private double length;
    private double width;

    public EllipseModified(Point2D start, double length, double width)
{
        this.start = start;
        this.length = length;
        this.width = width;
    }

    @Override
    public Point2D getStartPoint() {
        return start;
    }
    @Override
    public void draw() {
        System.out.println("Drawing Ellipse at " + start + " with
length " + length + " and width " + width);
    }
}
```

(**Canvas.java** on GitHub)

```
import javafx.geometry.Point2D;
import java.util.ArrayList;

public class Canvas {
    private ArrayList<Shape> shapes;

    public Canvas() {
        shapes = new ArrayList<>();
    }

    public void addShape(Shape shape) {
        shapes.add(shape);
    }

    public void removeShape(Shape shape) {
        shapes.remove(shape);
    }

    public Shape getShape(Point2D point) {
        Shape closestShape = null;
        double minDistance = Double.MAX_VALUE;
        for (Shape shape : shapes) {
            double distance = shape.getStartPoint().distance(point);
            if (distance < minDistance) {
                minDistance = distance;
                closestShape = shape;
            }
        }
        return closestShape;
    }

    public void drawAll() {
        for (Shape shape : shapes) {
            shape.draw();
        }
    }
}
```

(**Main.java** on GitHub)

```
import javafx.geometry.Point2D;

public class Main {
    public static void main(String[] args) {
        Canvas canvas = new Canvas();
        LineModified lineModified = new LineModified(new Point2D(0,
0), new Point2D(10, 10));
        RectangleModified rectangleModified = new
RectangleModified(new Point2D(5, 5), 2, 3);
        EllipseModified ellipseModified = new EllipseModified(new
Point2D(3, 3), 4, 2);

        canvas.addShape(lineModified);
        canvas.addShape(rectangleModified);
        canvas.addShape(ellipseModified);

        System.out.println("Drawing all shapes on the canvas:");
        canvas.drawAll();

        Point2D testPoint = new Point2D(4, 4);
        Shape closest = canvas.getShape(testPoint);
        System.out.println("\nThe closest shape to " + testPoint + "
is:");
        closest.draw();

        System.out.println("\nRemoving the closest shape and drawing
again:");
        canvas.removeShape(closest);
        canvas.drawAll();
    }
}
```

4.

(**Moveable.java** on GitHub)

```
public interface Moveable {
    void move(double dx, double dy);
}
```

(**Diamond.java** on GitHub)

```
import javafx.geometry.Point2D;

public class Diamond implements Shape, Moveable, Comparable<Diamond> {
    private Point2D start;
    private double length;
    private double width;

    public Diamond(Point2D start, double length, double width) {
        this.start = start;
        this.length = length;
        this.width = width;
    }

    @Override
    public void draw() {
        System.out.println("Drawing Diamond at " + start + " with
length " + length + " and width " + width);
    }

    @Override
    public Point2D getStartPoint() {
```

```java
        return start;
    }

    @Override
    public void move(double dx, double dy) {
        start = start.add(dx, dy);
    }

    @Override
    public int compareTo(Diamond other) {
        double myArea = this.length * this.width;
        double otherArea = other.length * other.width;
        return Double.compare(myArea, otherArea);
    }
}
```

(**DiamondDriver.java** on GitHub)

```java
import javafx.geometry.Point2D;
public class DiamondDriver {
    public static void main(String[] args) {
        Diamond diamond1 = new Diamond(new Point2D(1, 1), 4, 3);
        Diamond diamond2 = new Diamond(new Point2D(2, 2), 5, 2);

        System.out.println("Before moving:");
        diamond1.draw();

        diamond1.move(3, 3);

        System.out.println("After moving:");
        diamond1.draw();

        int comparisonResult = diamond1.compareTo(diamond2);
        if (comparisonResult > 0) {
            System.out.println("Diamond1 is larger than Diamond2.");
        } else if (comparisonResult < 0) {
            System.out.println("Diamond1 is smaller than Diamond2.");
        } else {
            System.out.println("Diamond1 and Diamond2 are of the same 
size.");
        }
    }
}
```

5. (**MoveableDemo.java** on GitHub)

```java
import javafx.geometry.Point2D;
import java.util.Arrays;

public class MoveableDemo {

    public static void main(String[] args) {
        int N = 5; // Example size of the array
        Moveable[] moveables = new Moveable[N];

        // Filling the array with Diamond objects
        moveables[0] = new Diamond(new Point2D(0, 0), 10, 5);
        moveables[1] = new Diamond(new Point2D(1, 1), 6, 4);
        moveables[2] = new Diamond(new Point2D(2, 2), 8, 3);
        moveables[3] = new Diamond(new Point2D(3, 3), 4, 7);
        moveables[4] = new Diamond(new Point2D(4, 4), 5, 5);

        // Moving each Diamond
        for (Moveable moveable : moveables) {
            moveable.move(1, 1); // Moving every diamond by an arbitrary
amount (dx=1, dy=1)
        }

        // Casting to Diamond[] for sorting (Arrays.sort() requires
Comparable[] type)
        Arrays.sort((Diamond[]) moveables);

        // Drawing each Diamond after sorting
        for (Moveable moveable : moveables) {
            ((Diamond) moveable).draw(); // Casting is required because
Moveable doesn't have the draw() method
        }
    }
}
```

This concludes Sheet (5) Solutions, this document + source code to all programming exercises available on https://bit.ly/CSE231Sheets .