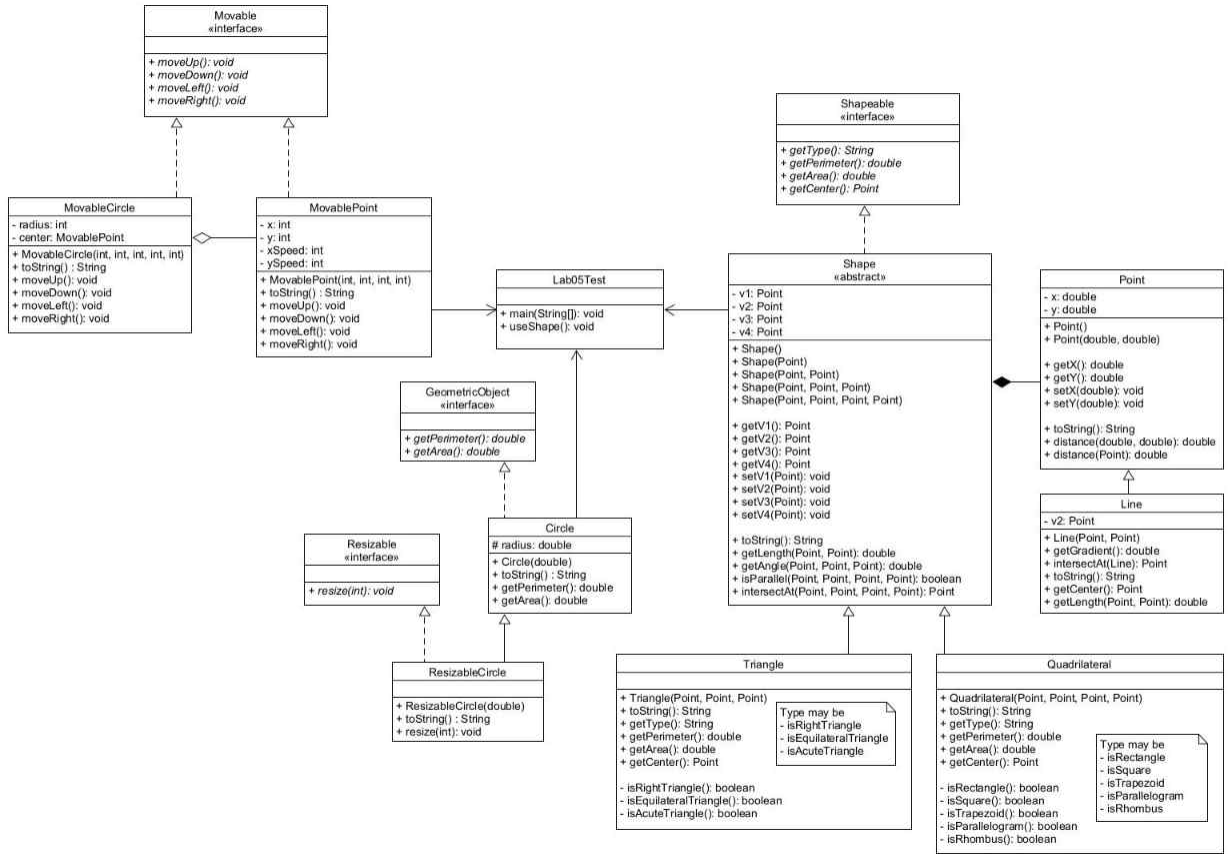
Java Programming (CSE220)

Lab 05

ID: 2020136129 Name: 최수연

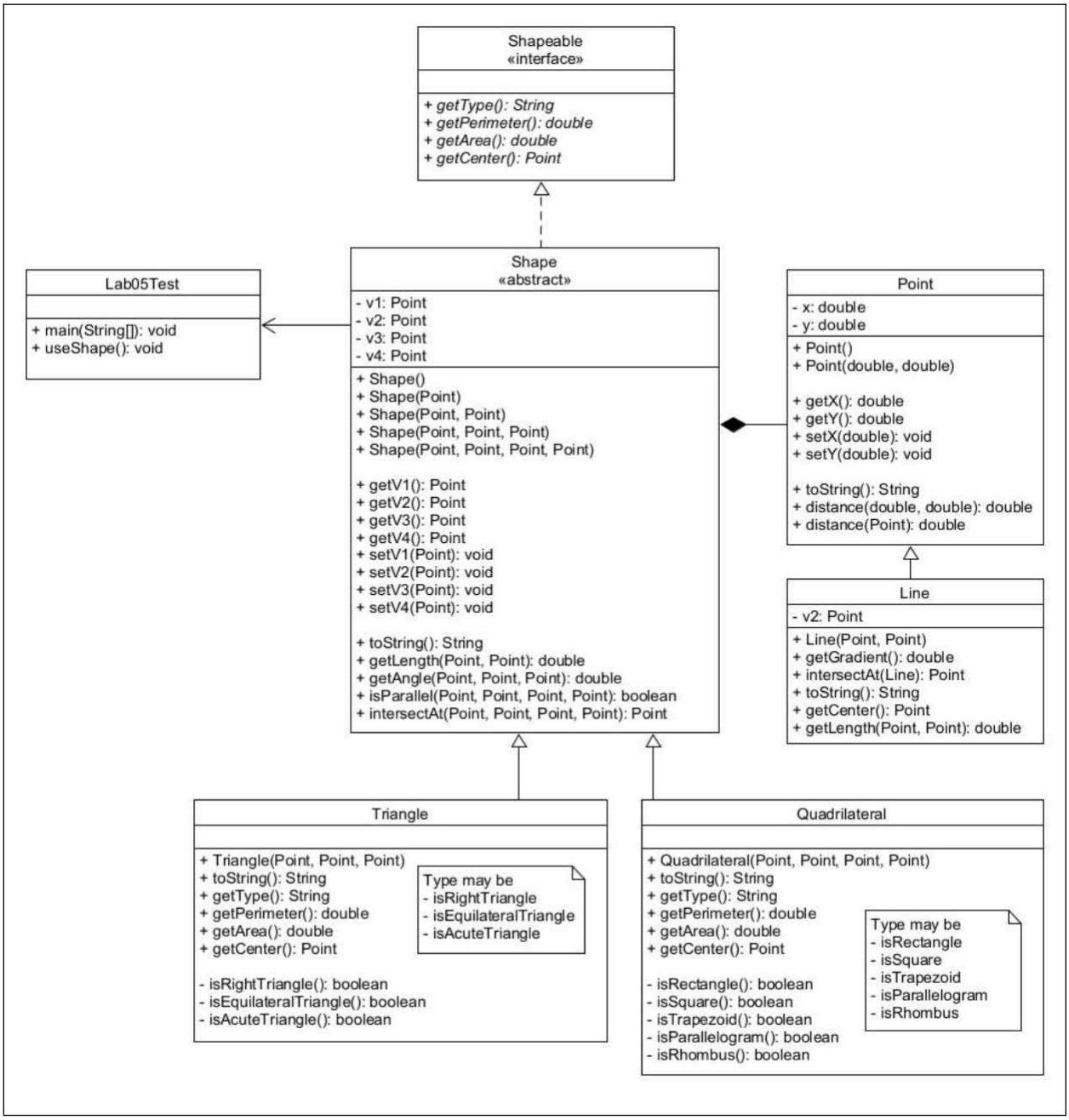
Class diagram



---

task-1: Consider the inheritance-based design for (Lab04 Task-1). Modify the design using **Polymorphism and Interfaces** and modify Java code accordingly.

Your diagram here



Code with Explanation

Lab05Test class

```
public class Lab05Test {

	public static void main(String[] args) {
		useShape();
	}

	public static void useShape() {
		// The four points
		Point v1 = new Point(0, 0);
		Point v2 = new Point(1, 1);
		Point v3 = new Point(2, 0);
		Point v4 = new Point(1, -1);

		// Line
		System.out.println("[Line]");
		Line l1 = new Line(v1, v2);
		System.out.println("1. " + l1);
		System.out.println("Gradient: " + l1.getGradient());
		System.out.println("Center: " + l1.getCenter());
		System.out.println();

		Line l2 = new Line(new Point(3, 3), new Point(1, 5));
		Point intersection1 = l1.intersectAt(l2);
		System.out.println("2. " + l2);

		if (intersection1 != null)
			System.out.println("Intersection between Line 1 and Line 2: " +
intersection1);
		else
			System.out.println("Line 1 and Line 2 are parallel.");
		System.out.println();

		Line l3 = new Line(new Point(3, 3), new Point(4, 4));
		Point intersection2 = l1.intersectAt(l3);
		System.out.println("3. " + l3);

		if (intersection2 != null)
			System.out.println("Intersection between Line 1 and Line 3: " +
intersection2);
		else
			System.out.println("Line 1 and Line 3 are parallel.");
		System.out.println();

		// Triangle
		System.out.println("[Triangle]");
		Triangle t1 = new Triangle(v1, v2, v3);
		System.out.println(t1);
		System.out.println("Type: " + t1.getType());
		System.out.println("Perimeter: " + t1.getPerimeter());
		System.out.println("Area: " + t1.getArea());
		System.out.println("Center: " + t1.getCenter());
```

```
            System.out.println();

            // Quadrilateral
            System.out.println("[Quadrilateral]");
            Quadrilateral q1 = new Quadrilateral(v1, v2, v3, v4);
            System.out.println(q1);
            System.out.println("Type: " + q1.getType());
            System.out.println("Perimeter: " + q1.getPerimeter());
            System.out.println("Area: " + q1.getArea());
            System.out.println("Center: " + q1.getCenter());
            System.out.println();

            // Distance between two points
            System.out.println("[Distance between two points]");
            System.out.println("v1 and v2 = " + v1.distance(v2));
            System.out.println("Line 1's v2 and Line 2's v2 = " +
l1.distance(l2));
      }
}
```

Shapeable interface

```
public interface Shapeable { // Define abstract methods
      String getType();
      double getPerimeter();
      double getArea();
      Point getCenter();
}
```

Point class

```
public class Point {
      private double x, y;

      public Point() {
            this.x = 0.0;
            this.y = 0.0;
      }

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

      public double getX() {
            return x;
      }

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

      public double getY() {
            return y;
      }
```

```
	public void setY(double y) {
		this.y = y;
	}

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

	public double distance(double x, double y) {
		// The distance between the current point and the point received by
the factor
		return Math.sqrt(Math.pow((this.x - x), 2) + Math.pow((this.y - y),
2));
	}

	public double distance(Point p) {
		// The distance between the current point and the point received by
the factor
		return Math.sqrt(Math.pow((this.x - p.x), 2) + Math.pow((this.y -
p.y), 2));
	}
}
```

Shape class

```
abstract public class Shape implements Shapeable {
	/**
	 * Mark as Abstract class because methods of Shapeable interface are not
implemented
	 * */
	private Point v1, v2, v3, v4;

	// Definition Shape constructors
	public Shape() {
		this.v1 = new Point();
		this.v2 = new Point();
		this.v3 = new Point();
		this.v4 = new Point();
	}

	public Shape(Point v1) {
		this.v1 = v1;
		this.v2 = new Point();
		this.v3 = new Point();
		this.v4 = new Point();
	}

	public Shape(Point v1, Point v2) {
		this.v1 = v1;
		this.v2 = v2;
		this.v3 = new Point();
		this.v4 = new Point();
	}

	public Shape(Point v1, Point v2, Point v3) {
```

```
		this.v1 = v1;
		this.v2 = v2;
		this.v3 = v3;
		this.v4 = new Point();
	}

	public Shape(Point v1, Point v2, Point v3, Point v4) {
		this.v1 = v1;
		this.v2 = v2;
		this.v3 = v3;
		this.v4 = v4;
	}

	public Point getV1() {
		return v1;
	}

	public void setV1(Point v1) {
		this.v1 = v1;
	}

	public Point getV2() {
		return v2;
	}

	public void setV2(Point v2) {
		this.v2 = v2;
	}

	public Point getV3() {
		return v3;
	}

	public void setV3(Point v3) {
		this.v3 = v3;
	}

	public Point getV4() {
		return v4;
	}

	public void setV4(Point v4) {
		this.v4 = v4;
	}

	@Override
	public String toString() {
		return "Shape [v1 = " + v1 + ", v2 = " + v2 + ", v3 = " + v3 + ", v4 =
" + v4 + "]";
	}

	public double getLength(Point v1, Point v2) {
		// distance between two points
		return Math.sqrt(Math.pow((v2.getX() - v1.getX()), 2) +
Math.pow((v2.getY() - v1.getY()), 2));
```

```
	}

	public double getAngle(Point v1, Point v2, Point v3) {
		// compute angle between two lines
		double m1 = (v2.getY() - v1.getY()) / (v2.getX() - v1.getX());
		double m2 = (v3.getY() - v2.getY()) / (v3.getX() - v2.getX());
		return Math.atan(Math.abs((m2 - m1) / (1 + m1 * m2))); // find Point
v2's angle
	}

	public boolean isParallel(Point v1, Point v2, Point v3, Point v4) {
		// whether the two lines are parallel
		// Same slope and different y-intercept
		double x1 = v1.getX(), y1 = v1.getY();
		double x2 = v2.getX(), y2 = v2.getY();
		double x3 = v3.getX(), y3 = v3.getY();
		double x4 = v4.getX(), y4 = v4.getY();

		double m12 = (y2 - y1) / (x2 - x1);
		double m34 = (y4 - y3) / (x4 - x3);

		double y12 = y1 - (m12 * x1);
		double y34 = y3 - (m34 * x3);

		double p = (x1 - x2) * (y3 - y4) - (y1 - y2) * (x3 - x4);

		if (m12 == m34 && y12 != y34 && p == 0)
			return true;
		else
			return false;
	}

	public Point intersectAt(Point v1, Point v2, Point v3, Point v4) {
		// calculate intersection
		double x1 = v1.getX(), y1 = v1.getY();
		double x2 = v2.getX(), y2 = v2.getY();
		double x3 = v3.getX(), y3 = v3.getY();
		double x4 = v4.getX(), y4 = v4.getY();

		double p = (x1 - x2) * (y3 - y4) - (y1 - y2) * (x3 - x4);
		double px = ((x1 * y2 - y1 * x2) * (x3 - x4) - (x1 - x2) * (x3 * y4 -
y3 * x4)) / p;
		double py = ((x1 * y2 - y1 * x2) * (y3 - y4) - (y1 - y2) * (x3 * y4 -
y3 * x4)) / p;

		if (p == 0) {
			System.out.println("parallel");
			return null;
		} else
			return new Point(px, py);
	}
}
```

Line class

```
public class Line extends Point { // Change Superclass from Shape to Point
```

```
	private Point v2; // Variable for point v2 is declared internally because multiple inheritances are not possible

	public Line(Point v1, Point v2) {
		super(v1.getX(), v1.getY()); // Only values for point v1 are defined as super classes
		this.v2 = v2;
	}

	public double getGradient() {
		// calculate the gradient : y2 - y1 / x2 - x1
		return (v2.getY() - this.getY()) / (v2.getX() - this.getX());
	}

	public Point intersectAt(Line l1) {
		// calculate intersection
		double x1 = this.getX(), y1 = this.getY();
		double x2 = v2.getX(), y2 = v2.getY();
		double lx1 = l1.getX(), ly1 = l1.getY();
		double lx2 = l1.v2.getX(), ly2 = l1.v2.getY();

		double p = (x1 - x2) * (ly1 - ly2) - (y1 - y2) * (lx1 - lx2);
		double px = ((x1 * y2 - y1 * x2) * (lx1 - lx2) - (x1 - x2) * (lx1 * ly2 - ly1 * lx2)) / p;
		double py = ((x1 * y2 - y1 * x2) * (ly1 - ly2) - (y1 - y2) * (lx1 * ly2 - ly1 * lx2)) / p;
		if (p == 0)
			return null;
		else
			return new Point(px, py);
	}

	@Override
	public String toString() {
		return "Line [v1 = (" + this.getX() + ", " + this.getY() + ") , v2 = (" + v2.getX() + ", " + v2.getY() + ")]";
	}

	public Point getCenter() {
		// get the center of a line
		double centerX = (this.getX() + v2.getX()) / 2;
		double centerY = (this.getY() + v2.getY()) / 2;
		return new Point(centerX, centerY);
	}

	public double getLength() { // Get this method from Shape class because this class does not inherit Shape class
		// distance between two points
		return Math.sqrt(Math.pow((v2.getX() - this.getX()), 2) + Math.pow((v2.getY() - this.getY()), 2));
	}
}
```

Triangle class

```
public class Triangle extends Shape {
```

```
	public Triangle(Point v1, Point v2, Point v3) {
		super(v1, v2, v3);
	}

	@Override
	public String toString() {
		return "Triangle [v1 = " + this.getV1() + ", v2 = " + this.getV2() +
", v3 = " + this.getV3() + "]";
	}
	@Override
	public String getType() {
		// output the type of a triangle
		if (isRightTriangle())
			return "It is a right-angled triangle.";
		else if (isEquilateralTriangle())
			return "It is a equilateral triangle.";
		else if (isAcuteTriangle())
			return "It is a acute-angled triangle.";
		else
			return "Nothing.";
	}
	@Override
	public double getPerimeter() {
		// get the perimeter of a triangle
		double a = getLength(this.getV1(), this.getV2());
		double b = getLength(this.getV2(), this.getV3());
		double c = getLength(this.getV3(), this.getV1());
		return a + b + c;
	}
	@Override
	public double getArea() {
		double a = getLength(this.getV1(), this.getV2());
		double b = getLength(this.getV2(), this.getV3());
		double c = getLength(this.getV3(), this.getV1());
		// Triangle by Heron's formula
		double s = (a + b + c) / 2.0;
		return Math.sqrt(s * (s - a) * (s - b) * (s - c));
	}
	@Override
	public Point getCenter() {
		// get the center of a triangle
		double centerX = (this.getV1().getX() + this.getV2().getX() +
this.getV3().getX()) / 3;
		double centerY = (this.getV1().getY() + this.getV2().getY() +
this.getV3().getY()) / 3;
		return new Point(centerX, centerY);
	}

	private boolean isRightTriangle() {
		// one angle of 90 degrees
		if ((getAngle(this.getV1(), this.getV2(), this.getV3()) == 90)
				|| (getAngle(this.getV2(), this.getV3(), this.getV1())
== 90)
				|| (getAngle(this.getV3(), this.getV1(), this.getV2())
```

```
== 90))
                        return true;
                else
                        return false;
        }

        private boolean isEquilateralTriangle() {
                // 3 equal sides & 3 equal angles
                if ((getAngle(this.getV1(), this.getV2(), this.getV3()) == 60)
                                && (getAngle(this.getV2(), this.getV3(), this.getV1())
== 60)
                                && (getAngle(this.getV3(), this.getV1(), this.getV2())
== 60)
                                && (getLength(this.getV1(), this.getV2()) ==
getLength(this.getV2(), this.getV3()))
                                && (getLength(this.getV2(), this.getV3()) ==
getLength(this.getV3(), this.getV1())))
                        return true;
                else
                        return false;
        }

        private boolean isAcuteTriangle() {
                // 3 angles all less than 90 degrees
                if ((getAngle(this.getV1(), this.getV2(), this.getV3()) < 90)
                                && (getAngle(this.getV2(), this.getV3(), this.getV1()) <
90)
                                && (getAngle(this.getV3(), this.getV1(), this.getV2()) <
90))
                        return true;
                else
                        return false;
        }
}
```

Quadrilateral class

```
public class Quadrilateral extends Shape {

        public Quadrilateral(Point v1, Point v2, Point v3, Point v4) {
                super(v1, v2, v3, v4);
        }

        @Override
        public String toString() {
                return "Quadrilateral [v1 = " + this.getV1() + ", v2 = " +
this.getV2() + ", v3 = " + this.getV3() + ", v4 = "
                                + this.getV4() + "]";
        }
        @Override
        public String getType() {
                // output the type of a quadrilateral
                if (isRectangle())
                        return "It is a rectangle.";
                else if (isSquare())
                        return "It is a square.";
```

```
		else if (isRhombus())
			return "It is a rhombus.";
		else if (isTrapezoid())
			return "It is a trapezium.";
		else if (isParallelogram())
			return "It is a parallelogram.";
		else
			return "Nothing.";
	}
	@Override
	public double getPerimeter() {
		// get the perimeter of a quadrilateral
		double a = getLength(this.getV1(), this.getV2());
		double b = getLength(this.getV2(), this.getV3());
		double c = getLength(this.getV3(), this.getV4());
		double d = getLength(this.getV4(), this.getV1());
		return a + b + c + d;
	}
	@Override
	public double getArea() {
		double a = getLength(this.getV1(), this.getV2());
		double b = getLength(this.getV2(), this.getV3());
		double c = getLength(this.getV3(), this.getV4());
		double d = getLength(this.getV4(), this.getV1());
		// Bretschneider's formula
		double s = (a + b + c + d) / 2.0;
		double theta = getAngle(this.getV1(), this.getV2(), this.getV3())
				+ getAngle(this.getV1(), this.getV4(), this.getV3());
		return Math.sqrt((s - a) * (s - b) * (s - c) * (s - d) - ((a * b * c *
d) * Math.pow(Math.cos(theta / 2), 2)));
	}
	@Override
	public Point getCenter() {
		// get the center of a quadrilateral
		double centerX = (this.getV1().getX() + this.getV2().getX() +
this.getV3().getX() + this.getV4().getX()) / 4;
		double centerY = (this.getV1().getY() + this.getV2().getY() +
this.getV3().getY() + this.getV4().getY()) / 4;
		return new Point(centerX, centerY);
	}

	private boolean isRectangle() {
		// opposite sides equal and parallel, 4 right angles
		if (isParallel(this.getV1(), this.getV2(), this.getV3(), this.getV4())
== true
				&& isParallel(this.getV1(), this.getV4(), this.getV3(),
this.getV2()) == true
				&& (getAngle(this.getV1(), this.getV2(), this.getV3())
== 90)
				&& (getAngle(this.getV2(), this.getV3(), this.getV4())
== 90)
				&& (getAngle(this.getV3(), this.getV4(), this.getV1())
== 90)
				&& (getAngle(this.getV4(), this.getV1(), this.getV2())
== 90)
```

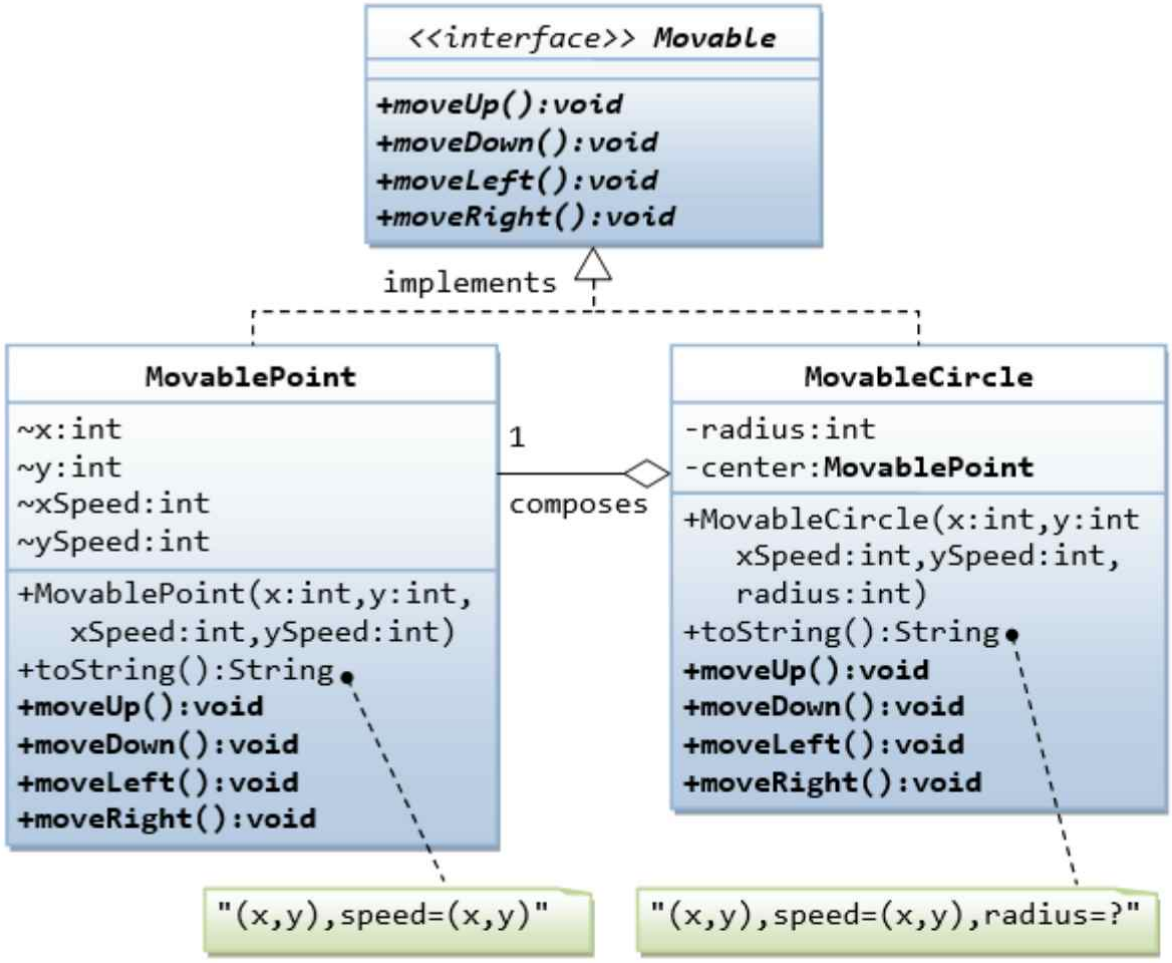
```
                        && (getLength(this.getV1(), this.getV2()) ==
getLength(this.getV3(), this.getV4()))
                        && (getLength(this.getV2(), this.getV3()) ==
getLength(this.getV4(), this.getV1())))
                return true;
            else
                return false;
        }

        private boolean isSquare() {
            // 4 equal sides, 4 right angles, opposite sides parallel
            if (isRectangle() == true && (getLength(this.getV1(), this.getV2()) ==
getLength(this.getV2(), this.getV3()))
                        && (getLength(this.getV3(), this.getV4()) ==
getLength(this.getV4(), this.getV1())))
                return true;
            else
                return false;
        }

        private boolean isTrapezoid() {
            // one pair of parallel sides
            if (isParallel(this.getV1(), this.getV2(), this.getV3(), this.getV4())
== true
                        || isParallel(this.getV1(), this.getV4(), this.getV3(),
this.getV2()) == true)
                return true;
            else
                return false;
        }

        private boolean isParallelogram() {
            // opposite sides equal and parallel, opposite angles equal
            if (isTrapezoid() == true
                        && (getLength(this.getV1(), this.getV2()) ==
getLength(this.getV3(), this.getV4()))
                        && (getLength(this.getV2(), this.getV3()) ==
getLength(this.getV4(), this.getV1())))
                return true;
            else
                return false;
        }

        private boolean isRhombus() {
            // 4 equal sides, opposite sides parallel, opposite angles equal
            if (isParallelogram() == true
                        && (getLength(this.getV1(), this.getV2()) ==
getLength(this.getV2(), this.getV3()))
                        && (getLength(this.getV3(), this.getV4()) ==
getLength(this.getV4(), this.getV1())))
                return true;
            else
                return false;
        }
}
```

Results/Output

```
[Line]
1. Line [v1 = (0.0, 0.0) , v2 = (1.0, 1.0)]
Gradient: 1.0
Center: (0.5, 0.5)

2. Line [v1 = (3.0, 3.0) , v2 = (1.0, 5.0)]
Intersection between Line 1 and Line 2: (3.0, 3.0)

3. Line [v1 = (3.0, 3.0) , v2 = (4.0, 4.0)]
Line 1 and Line 3 are parallel.

[Triangle]
Triangle [v1 = (0.0, 0.0), v2 = (1.0, 1.0), v3 = (2.0, 0.0)]
Type: It is a acute-angled triangle.
Perimeter: 4.82842712474619
Area: 0.9999999999999996
Center: (1.0, 0.3333333333333333)

[Quadrilateral]
Quadrilateral [v1 = (0.0, 0.0), v2 = (1.0, 1.0), v3 = (2.0, 0.0), v4 = (1.0, -1.0)]
Type: It is a rhombus.
Perimeter: 5.656854249492381
Area: 2.0
Center: (1.0, 0.0)

[Distance between two points]
v1 and v2 = 1.4142135623730951
Line 1's v2 and Line 2's v2 = 4.242640687119285
```

Task 2: Consider the following design:

Write an interface called **Movable**, which contains abstract methods **moveUp(), moveDown(), moveLeft() and moveRight()**, as shown in the class diagram. Also write the implementation classes called **MovablePoint and MovableCircle**. Mark all the overridden methods with annotation @Override.



Code with Explanation

Lab05Test class

```
public class Lab05Test {

	public static void main(String[] args) {
		useMovable();
	}

	public static void useMovable() {
		MovablePoint p1 = new MovablePoint(1, 2, 1, 1);
		System.out.println(p1);
```

```
            p1.moveDown();
            System.out.println(p1);
            p1.moveRight();
            System.out.println(p1);
            System.out.println("========================");

            // Test Polymorphism
            Movable p2 = new MovablePoint(3, 4, 1, 1); // upcast
            System.out.println(p2);
            p2.moveUp();
            System.out.println(p2);
            MovablePoint p3 = (MovablePoint) p2; // downcast
            System.out.println("downcast!");
            System.out.println(p3);
            System.out.println("========================");

            Movable m2 = new MovableCircle(1, 2, 3, 4, 20);  // upcast
            System.out.println(m2);
            m2.moveRight();
            System.out.println(m2);
            m2.moveUp();
            System.out.println(m2);
      }
}
```

Movable interface

```
public interface Movable {  // use keyword "interface" (instead of "class") to
define an interface
   // An interface defines a list of public abstract methods to be implemented by
the subclasses
   public void moveUp();    // "public" and "abstract" optional
   public void moveDown();
   public void moveLeft();
   public void moveRight();
}
```

MovablePoint class

```
/**
 * The subclass MovablePoint needs to implement all the abstract methods defined
 * in the interface Movable
 */
public class MovablePoint implements Movable {
      // Private member variables
      private int x, y, xSpeed, ySpeed; // x, y, xSpeed, ySpeed

      /** Constructs a MovablePoint instance at the given x and y */
      public MovablePoint(int x, int y, int xSpeed, int ySpeed) {
            this.x = x;
            this.y = y;
            this.xSpeed = xSpeed;
            this.ySpeed = ySpeed;
      }

      /** Returns a self-descriptive string */
```

```
	@Override
	public String toString() {
		return "point=(" + x + "," + y + "), speed=(" + this.xSpeed + "," +
this.ySpeed + ")";
	}

	// Need to implement all the abstract methods defined in the interface
Movable
	@Override
	public void moveUp() {
		y += ySpeed; // y-axis pointing up for 2D graphics
		System.out.println("up y-axis!");
	}

	@Override
	public void moveDown() {
		y -= ySpeed; // y-axis pointing down for 2D graphics
		System.out.println("down y-axis!");
	}

	@Override
	public void moveLeft() {
		x -= xSpeed; // x-axis pointing left for 2D graphics
		System.out.println("left x-axis!");
	}

	@Override
	public void moveRight() {
		x += xSpeed; // x-axis pointing right for 2D graphics
		System.out.println("right x-axis!");
	}
}
```

MovableCircle class

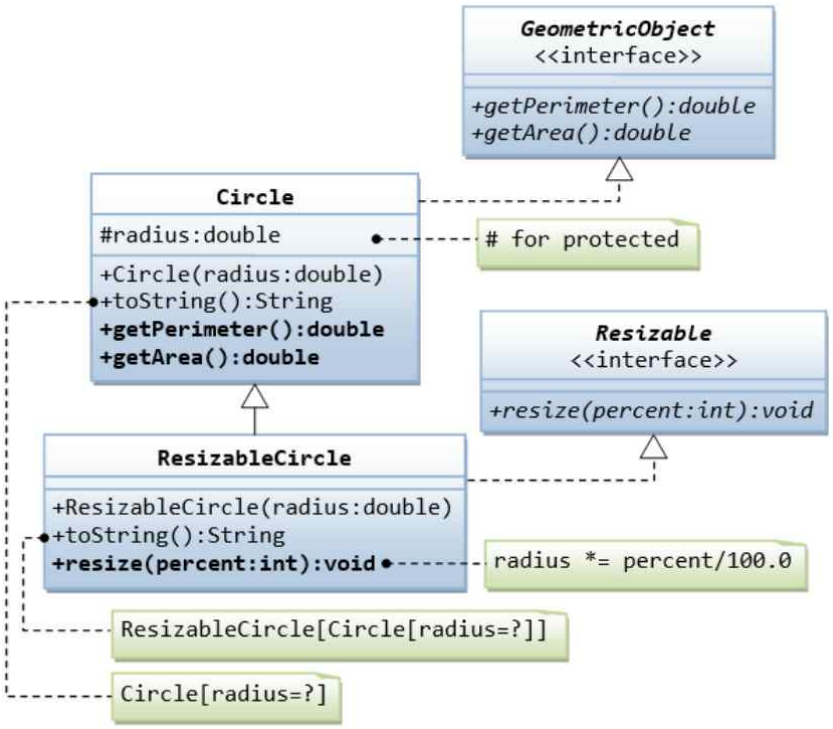
```
/**
 * The subclass MovableCircle needs to implement all the abstract methods
 * defined in the interface Movable
 */
public class MovableCircle implements Movable {
	private int radius;
	private MovablePoint center; // Use the instance of the existing class

	public MovableCircle(int x, int y, int xSpeed, int ySpeed, int radius) {
		// Create a new object of type 'MovablePoint'
		this.center = new MovablePoint(x, y, xSpeed, ySpeed);
		this.radius = radius;
	}

	/** Returns a self-descriptive string */
	@Override
	public String toString() {
		return center + ", radius=" + this.radius;
	}
```

```
	// Need to implement all the abstract methods defined in the interface
Movable
	@Override
	public void moveUp() {
		center.moveUp();
	}

	@Override
	public void moveDown() {
		center.moveDown();
	}

	@Override
	public void moveLeft() {
		center.moveLeft();
	}

	@Override
	public void moveRight() {
		center.moveRight();
	}
}
```

Results/Output

```
point=(1,2), speed=(1,1)
down y-axis!
point=(1,1), speed=(1,1)
right x-axis!
point=(2,1), speed=(1,1)
=======================
point=(3,4), speed=(1,1)
up y-axis!
point=(3,5), speed=(1,1)
downcast!
point=(3,5), speed=(1,1)
=======================
point=(1,2), speed=(3,4), radius=20
right x-axis!
point=(4,2), speed=(3,4), radius=20
up y-axis!
point=(4,6), speed=(3,4), radius=20
```

Task 3: Consider the following design



1. Write the interface called GeometricObject, which declares two abstract methods: getParameter() and getArea(), as specified in the class diagram.
2. Write the implementation class `Circle`, with a protected variable `radius`, which implements the interface `GeometricObject`.
3. The class `ResizableCircle` is defined as a subclass of the class `Circle`, which also implements an interface called `Resizable`, as shown in class diagram.
4. Write a test program to test Circle and `ResizableCircle classes.`

Code with Explanation

Lab05Test class

```
public class Lab05Test {

	public static void main(String[] args) {
		useCircle();
	}

	public static void useCircle() {
		Circle c1 = new Circle(4);
		System.out.println(c1);
		System.out.println("Perimeter: " + c1.getPerimeter());
		System.out.println("Area: " + c1.getArea());
		System.out.println();

		Circle c2 = new ResizableCircle(8); // upcast
```

```
                System.out.println(c2);
                System.out.println("Perimeter: " + c2.getPerimeter());
                System.out.println("Area: " + c2.getArea());
                System.out.println();

                System.out.println("downcast!");
                ResizableCircle c3 = (ResizableCircle) c2; // downcast
                System.out.println(c3);
                System.out.println("Resize!");
                c3.resize(40);
                System.out.println(c3);
                System.out.println();
        }
}
```

GeometricObject interface

```
public interface GeometricObject {
        double getPerimeter();
        double getArea();
}
```

Resizable interface

```
public interface Resizable {
        void resize(int percent);
}
```

Circle class

```
public class Circle implements GeometricObject {
        protected double radius;

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

        public String toString() {
                return "Circle[raduis=" + this.radius + "]";
        }

        // Implement methods defined in the interface GeometricObject
        @Override
        public double getPerimeter() {
                return 2 * Math.PI * radius;
        }

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

ResizableCircle class

```
public class ResizableCircle extends Circle implements Resizable {
        public ResizableCircle(double radius) {
                super(radius);
```

```
        }

        public String toString() {
                return "ResizableCircle[Circle[raduis=" + this.radius + "]]";
        }

        // Implement method defined in the interface Resizable
        @Override
        public void resize(int percent) { // resize radius
                this.radius *= percent / 100.0;
        }
}
```

Results/Output

```
Circle[raduis=4.0]
Perimeter: 25.132741228718345
Area: 50.26548245743669

ResizableCircle[Circle[raduis=8.0]]
Perimeter: 50.26548245743669
Area: 201.06192982974676

downcast!
ResizableCircle[Circle[raduis=8.0]]
Resize!
ResizableCircle[Circle[raduis=3.2]]
```

Conclusion

다형성과 인터페이스에 대한 수업 내용을 실습을 해보면서 복습할 수 있어서 이를 이해하는 데 도움이 많이 되었다. 원래는 자바가 계속 어려운 언어라는 생각에 사로잡혀 제대로 학습하려고 노력하지 않았던 것 같은데, 이번 수업과 강의자료를 통해 학습해보면서 쉽게 이해할 수 있었던 것 같았다.

It was helpful to understand this because I could review the contents of classes on polymorphism and interfaces while practicing. Originally, I didn't seem to have tried to learn properly because I was obsessed with the idea that Java was a difficult language, but it seemed that I could understand it easily while learning through this class and lecture materials.