# Lab 4

## Demo Points (20 marks)

1. Story 8: Git
    a. Local Repository (5 Marks)
    b. Remote Repository (5 Marks)
2. Story 9: Test Class
    a. Prints Point and Circle on Console (10 Marks)

## Next Week

Next Week you will start on a set of stories that will continue for two weeks. You will develop a hierarchy of shapes implementing methods for each. The UML diagram below shows the shapes to be developed.

The Quadrilateral is the base class. In particular, it is a convex quadrilateral so that the sum of all of the angles is 360 degrees. It contains four points: top loft, top right, bottom left and bottom right, which define the quadrilateral. These points represent the four corners of the all of the subclasses of the quadrilateral. It also has two methods: toString(), which is overridden from the Object class, and isRightShape(), which returns true if the object is a convex quadrilateral, false otherwise.

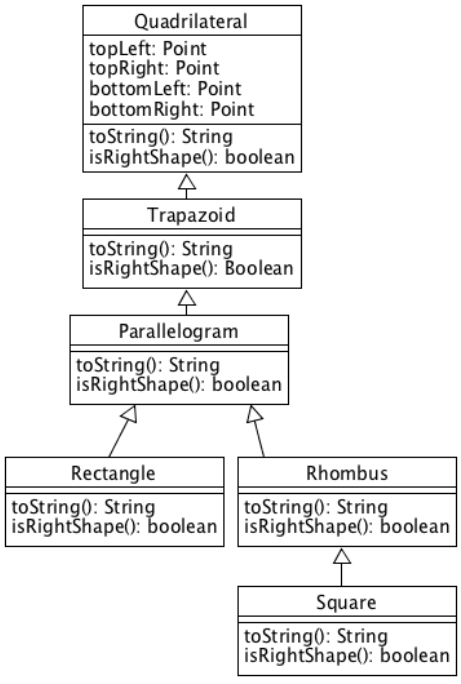


*Fig 1: Shapes Hierarchy in UML*

Subclasses override toString and isRightShape. Trapezoid (or Trapezium in UK english) is a convex quadrilateral that has two parallel lines. Testing for parallel lines can be done by checking that the sum of either the angles on the top and bottom of the quadrilateral are 180 degrees, or the sum of the right and left are 180 degrees.

The Parallelogram is a Trapezoid in which the top side parallels the bottom and the right side parallels the left. The Rectangle is a parallelogram in which all four angles are 90 degrees. The Rhombus is a parallelogram in which all of the sides are the same length and the square is a Rhombus in which all of the four angles are 90 degrees.

Below is some code you may find useful in developing the project. There is a class called Shapes that contains two static function: *distance*, which takes two point and calculates the euclidian distance between them, and *angle*, which takes three points and calculates the angle formed by them from the point of view of the first angle. In addition, there are test files for each of the classes to be written for this program.

## Shapes

You will be working in the same project in which you creates Circle and Point. First, you will add a class called Shapes to the src folder and the package you used for Circle and Point (e.g., oop.shapes). The Shapes class has the two utility functions defined above. The code for the utility functions is defined below. You may cut and paste them into your program.

```
public class Shapes {

	public static boolean compare(double a, double b) {
		return Math.abs(a - b) < .001;
	}

	public static double square(double a) {
		return a * a;
	}

	// Calculated the Euclidian distance between two points
	public static double distance(Point a, Point b) {
		return Math.sqrt(square(a.getX() - b.getX()) + square(a.getY() -
b.getY()));
	}

	// Calculates the angle from three points based on the law of cosines
	public static int angle(Point pointA, Point pointB, Point pointC) {
		double a = distance(pointA, pointB);
		double b = distance(pointA, pointC);
		double c = distance(pointB, pointC);
		double squares = square(a) + square(b) - square(c);
		double degrees = Math.toDegrees(Math.acos(squares / (2 * a * b)));
		return (int) Math.round(degrees);
	}

}
```

## Quadrilateral Test

You will create the classes Quadrilateral, Trapezoid, Parallelogram, Rhombus, Rectangle and Square. To do this first create a JUnit test with the name of the class followed by the word “Test” (for example “QuadrilateralTest”).

The Class QuadrilateralTest is a template for the tests for all the other classes. To create the other classes, add a JUnit test suite to you package in the test folder. Then, copy the functions into your new class. The functions define each of the shapes in such a way that they satisfy the test for the class under test but doe not satisfy the tests for subclasses. You may copy the code for the tests into the next test and change the assertion from “assertTrue” to “assertFalse” for those shapes that no longer match the new test. For example, as you can see below, for QuadrilateralTest, only testNotQuadrilateral asserts the isRightShape to be false; all the rest will assert the test to be true. For Trapezoid, testQuadrilateral will assert the test to be false (a quadrilateral is not necessarily a Trapezoid, but all the rest will assert the test to be true. For SquareTest, only testSquare will assert the test to be true; all the rest will assert it to be false.

To add a new class, first add the class (say Quadrilateral) to the src folder and the project package. Next, add the associated test to the test folder and the project package. Add code to the class until the test passes. Do this slowly. Add a line and test it, then add another line., You might consider added the tests to your test file one by one.

```
public class QuadrilateralTest {


	Point tl = new Point(0, 0);
	Point tr = new Point(0, 0);
	Point bl = new Point(0, 0);
	Point br = new Point(0, 0);
	Quadrilateral q;


	@Test
	public void testNotQuadrilateral() {
		//not Quadrilateral
		tl = new Point(0, 0);
		tr = new Point(0, 0);
		bl = new Point(0, 0);
		br = new Point(0, 0);
		q = new Quadrilateral(tl, tr, br, bl);
		assertFalse(q.isRightShape());
```

```
}

@Test
public void testConcaveQuadrilateral() {
      //not Quadrilateral
      tl = new Point(0, 4);
      tr = new Point(2, 2);
      bl = new Point(4, 4);
      br = new Point(2, 0);
      q = new Quadrilateral(tl, tr, br, bl);
      assertFalse(q.isRightShape());
}

@Test
public void testQuadrilateral() {
      //Quadrilateral
      tl = new Point(0, 1);
      tr = new Point(3, 2);
      bl = new Point(0, 0);
      br = new Point(4, 1);
      q = new Quadrilateral(tl, tr, bl, br);
      System.out.format("%n%s%n", q);
      assertTrue(q.isRightShape());
}

@Test
public void testTrapazoid() {
```

```
        //Trapezoid
        tl = new Point(1, 2);
        tr = new Point(2, 2);
        bl = new Point(0, 0);
        br = new Point(4, 0);
        q = new Quadrilateral(tl, tr, bl, br);
        assertTrue(q.isRightShape());
}

@Test
public void testParralelogram() {
        // Parallelogram
        tl = new Point(4, 3);
        tr = new Point(10, 3);
        bl = new Point(0, 0);
        br = new Point(6, 0);
        q = new Quadrilateral(tl, tr, bl, br);
        assertTrue(q.isRightShape());
}

@Test
public void testRhombus() {
        // Rhombus
        tl = new Point(4, 3);
        tr = new Point(8, 3);
        bl = new Point(0, 0);
        br = new Point(4, 0);
```

```
        q = new Quadrilateral(tl, tr, bl, br);
        assertTrue(q.isRightShape());
    }

    @Test
    public void testRectangle() {
        //Rectangle
        tl = new Point(0, 0);
        tr = new Point(0, 2);
        bl = new Point(1, 0);
        br = new Point(1, 2);
        q = new Quadrilateral(tl, tr, bl, br);
        assertTrue(q.isRightShape());
    }

    @Test
    public void testSquare() {
        //Square
        tl = new Point(0, 0);
        tr = new Point(0, 1);
        bl = new Point(1, 0);
        br = new Point(1, 1);
        q = new Quadrilateral(tl, tr, bl, br);
        assertTrue(q.isRightShape());
    }

}
```

## Next Week's (and the following week's) Marks

During the Demos for the next two weeks you will be working on the same set of stories from the backlog. I recommend that you demo at least the first story the first week. There is still a lot of work left, but the first time through will be the most difficult.

### Stories 9-27 (18 Stories) (72 marks)

For each story (4 Marks):

1. There are no compiler errors in the project (1 mark)
2. All of the JUnit tests for the previous stories work (1 mark)
3. The JUnit test runs for the current story (1 mark)
4. The story completes as shown in the example column of the backlog (1 mark)