Lab 8

Demo Points

1. Story 1 (1 Mark)
2. Strory 2 (2 Marks)
3. Story 3 (2 Marks)
4. Story 4 (3 Marks)
5. Story 5 (4 Marks)
6. Story 6 (3 Marks)
7. Story 7 (4 Marks) The tests must actually test the methods; not just succeed.
    1. Tests compile (2 marks)
    2. Tests run successfully (2 marks)
8. Story 8 (3 Marks)
9. Story 9 (4 Marks) The tests must actually test the methods; not just succeed.
    1. Tests compile (2 marks)
    2. Tests run successfully (2 marks)
10. Story 10 (4 Marks) The tests must actually test the methods; not just succeed.
    1. Tests compile (2 marks)
    2. Tests run successfully (2 marks)
11. Story 11 (4 Marks) The tests must actually test the methods; not just succeed.
    1. Tests compile (2 marks)
    2. Tests run successfully (2 marks)
12. Story 12 (4 Marks) The tests must actually test the methods; not just succeed.
    1. Tests compile (2 marks)
    2. Tests run successfully (2 marks)
13. Story 13 (2 Marks)

Next Week

Next week we will finish up the Shapes program by changing the constructor functions for the quadrilaterals. Now, they will throw exception rather than returning a Quadrilateral with all of the points set to 0. Return a particular object from a constructor when the constructor does not work is bad design. It requires anyone who uses the constructor to know what value represents a failure. Null is sometimes used as the special value, but returning null leads to null value exceptions. Generally, it is better to raise an exception when a constructor fails.

The first task will be to update the JUnit tests so they will catch a thrown exception whenever the constructor fails. They will also catch the exception when the constructor succeeds and immediately fail.

```
@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);
    try {
        q = new Quadrilateral(tl, tr, bl, br);
    } catch (ShapeException e) {
        assertEquals("Shape Error: "
                + "p(0, 0), p(0, 0), p(0, 0), p(0, 0) "
                + "does not form a quadrilateral",
                e.getMessage());
    }
    if (q != null) {
        fail("Should have thrown exception: " + tl + tr + br + bl);
    }
}
```

Figure 1: Example of a failing constructor

In the failing example, the constructor throws an exception which is caught. The test then checks that the exception has the right message. The test also fails if the constructor initializes the variable.

```
@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);
    try {
        q = new Quadrilateral(tl, tr, bl, br);
    } catch (ShapeException e) {
        fail(e.getMessage());
    }
    System.out.format("%n%s%n", q);
    assertTrue(q.isRightShape());
}
```

Figure 2: Example of a successful constructor

To test a successful constructor, you still need to catch the potential exception. Now, however, the test immediately fails when the exception is caught.

Because the Quadrilateral is the superclass of all of the other shapes, you will need to alter their tests to make the tests run because the constructor now throws an exception. You will need to catch this exception to make the project compile. As a work around, you can copy the original Quadrilateral class

into another class, says OldQuadrilateral and make it the new superclass Trapezoid. If, instead, you work through the errors, you will complete all of the tasks for the week.

Next Week's Demo Marks (30 Marks)

1. Story 32 (5 Marks)
2. Story 33 (5 Marks)
3. Story 34 (5 Marks)
4. Story 35 (5 Marks)
5. Story 36 (5 Marks)
6. Story 37 (5 Marks)