First Class: Points and Lines with Class Composition

CS 5004 Object Oriented Design

1. **Goals**

* Create a class from scratch
* Create a driver to test that class
* Create multiple types of constructors
* Create getters and setters for private variables
* Demonstrate data encapsulation with a setter test
* Reverse engineer an existing JUnit test file

ICE:

Your TA will describe this week’s assignment and then divide you up into groups. In your groups do not share code, but do

* Review the assignment itself
* Review the test file and briefly describe what each test is testing and what the test implies
* Discuss how to build a frame for this assignment
* Describe how composition will be used
* Build a frame by creating the files you are going to need in your IDE and loading the testing file
* Help each other pass the first two point tests:
  + @Test
  + **public** void noArgsPointConstructorTest() {
  + assertEquals(0,p1.getX() + p1.getY());
  + }
  + @Test
  + **public** void twoArgsPointConstructorTest()
  + {
  + assertEquals(1,p2.getX());
  + assertEquals(2,p2.getY());
  + }

If you finish all of this, go ahead and break off and feel free to get started with the reminder of the lab assignment.

Submit the names of each person in your group, the progress you made, and screenshots of passing the first two tests with your completed frame.

1. **Instructions:**

Testing File

For this assignment, I’ll give you the testing file instead of asking you to create it. A large part of your grade will be making sure you pass all of the tests in this file. Feel free to reverse engineer these tests to make sure you get it right, but don’t change any of the tests. If you add any tests, put them in their own section and make sure to add comments clearly identifying the tests you added.

<https://www.dropbox.com/s/pit0u437utetvji/PointLineTester.java?dl=0>

Part 1: Point Class

We will get started by creating a class to represent a cartesian point. Here are the requirements for this class.

1. Create a class called point made up of two integer values x and y.
2. Make sure x and y are private and include getters and setters for access.
3. Make sure the user can not set a value greater than 99 or less than -99. If they try, just set the value to 99/-99. Make sure this can’t be circumvented.
4. Create a no argument and a two argument constructor.
5. Create a method getQuadrant that will print the quadrant A, B, C, D, or origin that a point is in. If a point is on a border, example (-4, 0), then print “Border.”

If you create a point p1, then calling this method would be:

p1.getQuadrant();

Create a print method that will print the x and y coordinates of a point. A call to this method from point p1 would look like this:

p1.printPoint();

1. Make sure this class passes the provided PointTester class
2. Add JavaDoc Style comments for the class and for each method

Part 2 : Line Class

Class composition is when we use one class as part of another class. A line is made up of two points. We could represent this by having a line incorporate 4 integer values: x1, y1 & x2, y2, but this is redundant. Instead, we will use the already created point class objects to make a line. As long as the point class is in the same folder or the same package they will be accessible to each other.

1. Create a class called “Line” that’s made up of two points p1 and p2.
2. Create a constructor that will accept nothing and set two points at the origin.
3. Create a constructor that will accept two points.
4. Create a constructor that will accept 4 ints and use point constructors to set p1 & p2.
5. Create a print method that will print both points using the print point methods.
6. Create a getLength() method that returns the length of the line. Use the formula:

d = √[(x2 − x1)2 + (y2 − y1)2]

1. Make sure this class passes all provided LineTester tests
2. Add JavaDoc Style comments for the class and for each method

Part 3 : Driver

Test driven development is useful, but when you first start programming it is also important for you to understand how your classes could be used. A driver class is a class object that builds a usable application instead of representing data. It is the class where your public static void main(...) method is located and where control flow will start. For some of the applications you create, I will ask you to also create a driver. This is such an assignment.

1. Create a driver that will generate a point in each quadrant and the origin.
2. Write manual test cases for each quadrant. For example:

Point p1 = new Point(0,0);

If (p1.getQuadrant() == “Origin”)

System.out.println(“Origin test passed”);

else System.out.println(“Origin test failed”);

1. Print each point to see if your print method works as expected
2. Create three lines and print each to see if the line print method works as expected
3. Check the line length of each created line and see if it produces expected results
4. Make sure to explain in comments what you are doing.

**3. Extensions:**

Outside academics you will not get specific requirements. Each lab assignment is worth 100 points, but the base requirements will only get you to 85% - 90%. If you want an A, you’ll have to find a way to go above and beyond what is asked. I’ll often make some suggestions to you in this section, but it is entirely up to you what you’d like to add to the assignment. Make sure you know who your grader is and discuss extension expectations with them. You won’t have to do all of the extensions to get credit.

Extension suggestions:

1. Add a more complex class like a square made up of lines. Don’t forget to test it.
2. Increase the percent coverage of the provided testing class.
3. Add more simple functionality to the points or lines like getting line slope.

SwapXY for point class, getSlope for line class

1. Add more complex functionality to the line class like checking to see if a line crosses another line.

Add check cross

1. Look ahead and figure out how to do a string override.

Overridden the toString method for point class

1. Look ahead and figure out how to do an equals override.

Overridden the equals and hashcode method for point class

1. Add additional functionality and add tests covering any functionality you added.

**4. Report:**

Each assignment must include a short report. The generation of this report should take you no more than 15 minutes. This gives you a chance to reflect back on what you learned and it makes grading easier on your grader. For this report, I want the following sections:

1. Reflection (*What did you learn? How did you feel about working from a test file?)*
2. Testing Comparison (What’s the difference between testing with JUnit tests and with a Driver? Why did I ask you to create a driver this time?)
3. Extensions (*What extensions are you requesting?)*
4. Grading Statement (*Based on the rubric, what grade do you feel you deserve? Be honest.)*

**5. Submission:**

Submit your files as a single zip file named: “Your Name”\_”Assignment”.zip

Unless your grader requests it, do not submit your entire project folder. Do not include any JavaDocs, and make sure you remove any package statements.

* Point.java
* Line.java
* PointLineTester.java
* Your report
* Add any additional files you need for your extensions

Submission checklist:

* Did you include all files requested?
* Did you include adequate comments?
* Did you include comment blocks at the top of each file?
* Did you name your files as requested?
* Does your code compile?
* Did you remove any package lines generated by your IDE?
* Did you take care of any warnings presented by your IDE?

|  |  |  |
| --- | --- | --- |
|  | **Possible** | **Given** |
| Tests | | |
| Provided Point Tests Pass (2 pts per test) | 26 | 0 |
| Provided Line Tests Pass (4 pts per test) | 16 | 0 |
| Point Class | | |
| Print Method Created (Driver Test) | 5 | 0 |
| Line Class | | |
| Line : Print method created (Driver test) | 5 | 0 |
| Driver | | |
| Point tested as requested | 10 | 0 |
| Line tested as requested | 10 | 0 |
| Misc | | |
| Report | 5 | 0 |
| Code Quality (correct indentation, comment blocks, variable naming, etc) | 10 | 0 |
| Not included in total possible: | | |
| Does not compile | -100 | 0 |
| Extensions (Not calculated without report) | 15 | 0 |
| Late penalty | -20 | 0 |
| Creative or went above and beyond | 10 | 0 |
| Code contains warnings | -20 | 0 |
|  | |  |
| TOTAL POINTS POSSIBLE out of 100 | 87 | 0 |