COMPSCI230S1C 2016 Assignment 3

Due date: Thursday 20 May 2016 at 4 pm.

Submit via the assignment dropbox (https://adb.auckland.ac.nz).

Your lecturer: Ulrich Speidel (ulrich@cs.auckland.ac.nz)

What you will learn in this assignment

In this assignment, you will design JUnit tests for two Java classes and run them in Eclipse. You will also be exposed to exception classes and exception handling in Java.

Getting prepared

1) Review the lecture notes to date
2) Download the zip file with the Java source files
3) Start Eclipse and create an Eclipse project (File -> New -> Java Project, the choose a project name and hit Finish)
4) Import the sources from the zip file into Eclipse

The walkabout

The Java code for this assignment is not an application. Rather, it consists of:

- Two Java classes (CoordinatePoint and Navigator) which you will develop tests for. These classes implement code that may be used for navigation purposes in a GPS or mapping application.
- Three Java classes that define exceptions used by the above.
- A sample class TestCoordinatePoint with a number of JUnit tests to give you an idea as to what is expected. In the first part, you will add to this class.
- A skeleton class TestNavigator which you will implement the majority of your tests in.
- OK, I've added an application (NavigatorApp) to show how Navigator might be used. Only if you're nosy!

In this assignment, we will largely use black box testing. In practice, this means that you must design your tests based only on the following information (unless specified otherwise):

- The public methods of each class (including constructors)
- The Javadoc comments for these methods
- Parameter and return types for these methods

The internal implementation of the classes must not lead you to include or exclude a test (e.g., you must not say things such as "method x() uses method y() internally, so if I test method x() then that automatically tests method y() as well".

The classes

CoordinatePoint

This class stores the coordinates of a point on the surface of the earth as a pair of two (double) values: longitude and latitude. The longitude (measured in degrees) can be between -180 and +180 degrees inclusive (negative values for the western hemisphere, positive values for the eastern hemisphere). Latitudes can be between -90 (South Pole) and +90 (North Pole). 0 degrees longitude are the Greenwich meridian and 0 degrees latitude denote the equator. For example, the entrance to the Owen G Glenn Building on Grafton Road is roughly at +174.77095 degrees longitude and

-36.85292 degrees latitude. If you are unfamiliar with any of these concepts, please familiarise yourself with these first (e.g., https://en.wikipedia.org/wiki/Geographic_coordinate_system).

N.B.: In this assignment, we will express longitude and latitude as degrees and fractions of whole degrees rather than as degrees, minutes, and seconds (which is also common). That is, 10 degrees and 45 minutes becomes 10.75 degrees for the purposes of this assignment etc. We also don't worry about altitudes here.

Note that the version of the CoordinatePoint class that is distributed with the assignment *does* contain faults. It is not your task to fix these faults, and you must not submit this class as part of your assignment. It is provided to you to assist you in the design of your tests. However, you may (and are in fact encouraged) to modify the code in the class to fix these faults or introduce other faults in order to "test your tests".

Navigator

This class uses CoordinatePoint objects to store an origin position and a destination position. If you look at its methods, you'll see that they also let us set the other origin's and destination's coordinates individually, so we don't necessarily have to handle (or even know about) CoordinatePoint objects ourselves in order to use the Navigator class.

The main purpose of the class is that it lets us compute the distance between the origin and the destination. This happens in the method distanceInKm(), which uses the Haversine formula (https://en.wikipedia.org/wiki/Haversine_formula) to compute the distance. The value 6371 that you see in its implementation is the radius of the earth (i.e., the distance from the centre of the earth to its surface). The method distanceInNM() does the same but the result is meant to be in nautical miles. One nautical mile is 1.852 km.

Note that the version of the Navigator class that is distributed with the assignment *may* contain faults. It is not your task to fix these faults, if any, and you must not submit this class as part of your assignment. It is provided to you to assist you in the design of your tests. However, you may (and are in fact encouraged) to modify the code in the class to fix any faults or introduce other faults in order to "test your tests".

CoordinateOutOfBoundsException

This abstract class extends Exception and serves as the superclass for the two custom exception classes LongitudeOutOfBoundsException and LatitudeOutOfBoundsException. This class does not require testing for the purposes of this assignment, and must not be submitted.

LongitudeOutOfBoundsException and LatitudeOutOfBoundsException

These classes implement exceptions that are to be thrown when we attempt to set a longitude or latitude to an illegal value (less than -180 degrees or more than +180 degrees for longitudes, less than -90 degrees or more than +90 degrees for latitudes). These classes themselves do not require testing for the purposes of this assignment (however their use by CoordinatePoint and Navigator does), and must not be submitted.

TestCoordinatePoint

This class contains a few tests already, to give you an idea of what's expected. Its purpose is to test the constructors and methods of the CoordinatePoint class. In part 1, you will add tests to this class, and you will need to submit your modified version of this class.

TestNavigator

This class is supplied as an empty shell. In part 2, you will add tests to this class in order to test the constructor and methods of the Navigator class. You will need to submit your modified version of this class.

NavigatorApp

This class is a sample program showing how you might use the Navigator class. This class is provided for your information only. No tests need to be written for it, and it must not be submitted as part of this assignment.

Important

Make a backup of your Java source code after each part and keep it. If for whatever reason, you are not able to complete, we'll be able to give you marks for the parts that you completed (rather than having to deal with a part that doesn't compile!).

How we will mark your assignment

Your submission will essentially consist of commented tests. We will run these tests against a variety of versions of CoordinatePoint and Navigator to which we will add a number of faults. All of these faults will allow CoordinatePoint and Navigator to compile. Your marks will be based on:

- Whether your tests that are commented so that the purpose of each test is obvious
- Whether **<u>your</u>** tests flush out the faults we introduce into CoordinatePoint and Navigator
- Whether your tests pass a correct implementation of CoordinatePoint and Navigator
- Whether your tests correctly handle any CoordinatePointOutOfBoundsException raised
- Whether your tests unnecessarily duplicate testing effort (within reason)

Part 1 - Testing the CoordinatePoint class - 20 marks

The present tests in TestCoordinatePoint test the constructors and getters of the class only - they do not test any other methods. What other tests are sensible to establish correct functioning of the class (assuming you can't see inside its methods to see how they're implemented)? Add tests that will flush out any faults in the class. Name your tests as test<method><number>. E.g., for method x(), name the tests testx1(), testx2() etc.

Tests should be specific to a particular functionality and not lump several functionalities into a single test, unless this is absolutely necessary to conduct the test. As a guide, each test should contain no more than at most two asserts and one fail. Look at the existing tests as a guide. You must not implement more than 10 tests in this part (an adequate solution is possible with fewer than 10 tests).

You must comment each test to explain its specific purpose in one or two lines. Do not modify or remove the existing tests or comments.

Submit your version of TestCoordinatePoint.java.

Part 2 - Testing the Navigator class - 80 marks

Now apply the skills you've learned for CoordinatePoint to the Navigator class. Again, assume that you don't know the implementation of the public methods except for method signature, return value, and type of any exceptions thrown. *The one exemption from this rule is the distanceInKm() method - here you are allowed to look inside and consider which faults are likely to occur due to* *sloppiness or minor oversights (e.g., using a sine function instead of a cosine, +/- swaps, using the wrong variables) and how you might be able to detect those.*

Consider which tests are most likely to cause failures as a result of faults in the implementations of constructor and methods in Navigator. Name your tests as test<method><number>. E.g., for method x(), name the tests testx1(), testx2() etc.

You must not write more than 60 tests for this class. You must comment each test to explain its specific purpose in one or two lines.

Submit your version of Navigator.java.

Submission

Submit all your TestCoordinatePoint.java and TestNavigator.java files via the assignment dropbox (https://adb.auckland.ac.nz). Upon submission, ensure that you submit the test files you have modified, rather than the originals. On all submissions, include any files you have submitted earlier even if they have not changed.

Note: Students requesting extensions for any reason will significantly support their cause by submitting at least part 1 (TestCoordinatePoint.java) by May 12, 2016.