CS 5010: Programming Design Paradigms

Fall 2017

HW 1 – Environment Setup and Review of Java

Assigned: Saturday, September 9, 2017, Due: Monday, September 18, 2017 by 6pm

College of Computer and Information Science
Northeastern University – Seattle

The purpose of this assignment is to help you set up your development environment, get reacquainted with Java, and start getting familiar with the tools that you will use for the rest of this course. Although this assignment's description is long, we expect that the step-by-step instructions will make doing the assignment less overwhelming than reading it may be.

Note: You will need to know this assignment's material for the remainder of the semester, so make sure that you understand thoroughly the tools and concepts in it. If you experience troubles with finishing on time, get in touch with the staff immediately so that we can help you get back on track.

Setup Problem 1 – Setting Up a Development Environment

Your very first assignment is to decide what computer(s) and development environment(s) you will use for this course (or at least for this assignment). We strongly recommend using the IntelliJ IDEA IDE in this course, but you are welcome to use any development environment that you are comfortable with. Note, however, that we going to support only IntelliJ IDEA.

Setup Problem 2 – Setting Up CCIS Git Access

Throughout the course, you will receive starter code, and you will submit your assignments through the CCIS's GitHub server (git). git is a version control system that lets software engineers backup, manage, and collaborate on large software projects.

For this problem, you should follow the setup instructions already posted on the course website, and linked here again for your convenience:

- Lesson 0.5: Introduction to Git
- Do the exercises in the Git Laboratory
- Lesson 0.6: Dealing with Conflicts in Git
- **CCIS Account:** Sign up for a CCIS account. You will not be able to submit assignments until you have a CCIS account, and you will need that account for GitHub access as well.
- Git and GitHub. Make sure git is installed on your computer.

You should familiarize yourself with these commands as you will need to use them throughout this homework, and the rest of the course.

Setup Problem 3 – Setting Up Maven

Throughout this course, you will also use Apache Maven built management tool.

In this problem, your task is to familiarize yourselves with Apache Maven, and set it up. Instructions on how to appropriately setup Maven have already been posted on the course website, under the link Maven Setup.

Problem 4 – Your First Java Class

For some reason, which she can't quite remember now, our imaginary student, Lydia, has decided to take four different courses this term, and now she has four different homework assignments to work on every week. But there is only so much time in a day, and she decides to write a code (program) that will randomly choose which homework should Lydia tackle first.

Please help Lydia write her program. In doing so, please create a Java class with a `main` method that will randomly choose, and then print to the console, one of the four possible course names that you yourselves define. Create the file `RandomHomework.java`, which will define a Java class `RandomHomework`. Your class `RandomHomework` will reside in the Java package `Assignment1`.

Hint 1: Java requires every runnable class to contain a `main` method, whose signature is public static `void main(String[] args)`. A sample code skeleton for the `RandomHomework` class is shown below. It is meant only as a starting point, and you are free to organize your code as you see fit.

```
/**
 * RandomHomework selects a random course name to display to the user.
 */
public class RandomHomework {

    /**
     * Uses a RandomHomework object to print
     * a random course name to the console.
     */
    public static void main(String[] argv) {
        RandomHomework randomHomework = new RandomHomewok();
        System.out.println(randomHomework.getCourse());
    }

    /**
     * @return a random course name from a list of four possible courses.
     */
    public String getCourse() {
        // YOUR CODE GOES HERE
    }
}
```

Hint 2 – Using java.util.Random: Please do not write your own random number generator to decide which course to select. Instead, take advantage of Java's `Random` class. More specifically, you probably want to use the `nextInt(int n)` method to choose your course. You can (should) read the documentation for `Random's nextInt(int n)` method by going to the Java API, and selecting `Random` from the list of classes. One way to choose a random course is using an array. This approach might look something like:

```
String[] course = new String[4];
course[0] = "Programming Design Paradigms";
course[1] = "Object Oriented Design";
course[2] = "Data Structures and Algorithms";
course[3] = "Data Mining";
```

Hint 3: Now would be a good idea to add your new class to version control, and commit your code.

What to turn in: As a solution to Problem 4, you will want to submit:

- Source code for the class `Assignment1/RandomCourse.java` that prints out one of four random course names when its main method is executed.
- A short description (one paragraph or less) of what you did in Problem 4. You can store your written description into a file **writeup.txt**.

Problem 5 – Testing Java Code with JUnit

Testing is an essential part of writing quality software, so it is important for testing to be a convenient part of software development. JUnit is a framework for creating unit tests in Java. A unit test is a test for checking that a given method in a class conforms to its specification for an input. This problem provides a quick overview, and simple example of how JUnit works. (Unit testing will be more significant in later assignments.)

Open both `Assignment1/Fibonacci.java` and `Assignment1/test/FibonacciTest.java` (from the comments, you can see that `FibonacciTest` is a test of the `Fibonacci` class).

Now run the JUnit test `Assignment1.test.FibonacciTest`. A window or panel with a menacing red bar will appear, indicating that some of the tests in `FibonacciTest` did not complete successfully. You should see the list of tests that failed, as well as a more detailed Failure Trace for the highlighted test. The first line in the Failure Trace should display an error message that explains why the test failed. (The author of the test code creates this error message.)

If you click on the failure `testThrowsIllegalArgumentException`, you should see the appropriate error message. In this example, the first line of the failure trace shows that `Fibonacci.java` improperly threw an `IllegalArgumentException` when tested with zero as its argument. If you double-click on the name of a test in the top pane, it will jump to the line where the failure occurred in the editor pane. Figure out the problem in `Fibonacci.java`, fix it, and rerun the JUnit test. After you have fixed all the problems in `Fibonacci.java`, you should see a bright green bar instead of a red one when you run `FibonacciTest`.

What to turn in: A successful solution to Problem 5 will consist of:

- A source code of a **modified** class `Assignment1/Fibonacci.java`, that passes the three JUnit tests. (Note that you should not edit `Assignment1/test/FibonacciTest.java` to accomplish this task.)
- A short description (one paragraph or less) of what you did in Problem 5, to modify the class Fibonacci.java. You can continue storing your written description into a file **writeup.txt**.

Problem 6 – Writing Your Own JUnit

In this problem, our imaginary student, Lydia, is busy again. One of her course assignments asked her to write a program that receives an integer input from a user, then reverses it, and prints the reversed number to the console. For example, if the input into a program is a number 25, then the output to the console should be 52.

Lydia think that she knows what to do, and she starts by creating a Java class `ReverseNumber`. In doing so, she uses the `java.util.Scanner` class. Her code is pretty straightforward, but unfortunately, when she tries to use it, it doesn't seem to work correctly.

Please help Lydia, and write your own version of the Java class `ReverseNumber` (still, using `java.util.Scanner class`). After that, please write a corresponding JUnit `ReverseNumberTest.java` to test `ReverseNumber` class.

Hint 1: Your `ReverseNumberTest.java` should test at least that:

- Class `ReverseNumber` throws an appropriate exception if no input number is given.
- Class `ReverseNumber` throws an appropriate exception if input number is in the wrong format.

What to turn in: A successful solution to Problem 6 will consist of:

- A source code for `Assignment1/ReverseNumber.java`.

- A source code for JUnit test `Assignment/ReverseNumberTest.java`.
- A short description (a paragraph or two, but not more that that) of how did you go about implementing `ReverseNumber.java`, and what did you test for in `ReverseNumberTest.java`. You can continue storing your written description into a file **writeup.txt**.

Problem 7 – Getting a Real Taste of Java – Vehicles on a Road

Until now, we have only been introducing tools. This problem dives a bit deeper into a real programming exercise. This problem may be a bit more challenging for some of you. If that is the case, please don't be discouraged. We are here to help, and we expect that time spent now will pay off significantly during the rest of the course.

Warm-Up: Creating a Vehicle: Take a look at a class `Vehicle.java`. A Vehicle is a simple object that has a velocity, expressed as a double, and a direction, expressed as an integer, where 1 denotes the Eastbound direction, and 2 the Westbound direction.

Is the class `Vehicle.java` functional? If not, fix all of the problems with it, and document your work.

Hint 1: In doing so, you may want to develop a JUnit class `VehicleTest.java` to help you with testing. Your unit test should test at least that:

- Class `Vehicle` throws an appropriate exception if no inputs are provided for the constructor.
- Class `Vehicle` throws an appropriate exception if inputs provided to the constructor have an incorrect format, or value.

Using Pre-Defined Data Structures: Next, create a class `Highway`. As before, a skeleton code is provided (see `Highway.java`). A `Highway` is a container for `Vehicles`. `Highway` should support all of the methods listed below, and it is your task to fill in the code to implement these methods correctly:

- `add(Vehicle)`
- `remove(Vehicle)`
- `getVelocityEastbound()`
- `getVelocityWestbound()`
- `numberVehiclesEastbound()`
- `numberVehiclesWestbound()`
- `contains(Vehicles)`

The specifications for these methods are found in the javadoc file for `Highway`. In `Highway`, we use a `java.util.Set` to keep track of the `Vehicles`. This is a great example of using a predefined Java data-structure to save yourself significant work. Before implementing each method, read the documentation for `Set`. Some of your methods will be as simple as calling the appropriate predefined methods for `Set`.

Hint 2: Before you start coding, please take time to think about the following questions. There seem to be at least two approaches to implementing functions getVelocityEastbound()/getVelocityWestbound():

- Every time function `getVelocityEastbound()`, or `getVelocityWestbound()` is called, go through all the vehicles on the `Highway`, determine which direction they are moving in, and then for the asked direction, find the velocity of the slowest `Vehicle`.
- Keep track of the slowest velocity in the eastbound direction, as well as in the westbound direction, and update those variables accordingly every time a new `Vehicle` is added or deleted. This eliminates the need to perform any computation when `getVelocityEastbond()/getVelocityWestbound()` is called.

Which approach do you think is better, and why?

Hint 3: Can a similar reasoning be applied to functions `numberVehiclesEastbound()` and `numberVehiclesWestbounds()`? Why, or why not?

Implementing a Scenic Road: In this problem, you will implement a class `ScenicRoad.java`, where `ScenicRoad` is also a container for `Vehicles`. The key difference between a `ScenicRoad` and a `Highway` is that a `ScenicRoad` typically has a significantly lower bandwidth than a highway, and as such, can take on a significantly smaller number of vehicles at once. Once there are too many vehicles on a `ScenicRoad`, we have a traffic jam, and the velocity of all of the vehicles on the road typically decreases to less than 5mph. The maximum number of `Vehicles` in a single direction (bandwidth) that a `ScenicRoad` can take on without getting jammed is defined when the constructor is called:

```
public ScenicRoad(int bandwidht)}};
```

Since a `ScenicRoad` is in many ways similar to a `Highway`, we internally keep track of many things in the `ScenicRoad` with a `Highway`, allowing us to reuse code. Many of the methods in `ScenicRoad` can simply delegate to the equivalent in `Highway`. This design of having one class contain an object of another class, and reusing many of the methods is called composition.

In addition to the constructor described above, you will need to implement the following new method in the `ScenicRoad`:

- `add(Vehicle)`

In doing so, keep in mind that too many `Vehicles` can create a traffic jam on a `ScenicRoad`, which, in turns, causes all `Vehicles` to slow down to 5mph, or less.
A few things to consider before you start writing code:

- If you will need to use sorting, you shouldn't implement your own sorting algorithm. Instead, take advantage of the Java API (remember: "Know and Use the Libraries").
- Also, you shouldn't change your implementation of `Highway` or `Vehicle` for this problem. In particular, you should not implement the Comparable interface. If you are tempted to do so, consider using `Comparator` instead. `Comparator` is a companion interface to Comparable, and is used throughout the Java libraries: check out the sort methods in `java.util.Collections` as an example.
- If you do make any changes to `Highway` or `Vehicle` for this problem, then explicitly document what changes you made and why.
- Be cautious if you plan on using Java's `TreeSet`; remember that `TreeSet` does not store duplicates, and if you provide a `TreeSet` with a `Comparator`, it will use that `Comparator` to determine duplication. See the `TreeSet` API documentation for more details.
- Writing a JUnit test should help you here.
- And, naturally, don't forget to commit your code more than occasionally.

What to turn in: A successful solution to Problem 7 will consist of:

- A source code for classes:
  - `Assignment1/Vehicle.java`,
  - `Assignment1/Highway.java`, and
  - `Assignment1/ScenicRoad.java`

  that pass their respective JUnit tests.
- A source code for JUnit test `Assignment1/VehicleTest.java, Assignemnt1.HighwayTest.java`, and `Assignment1.ScenicRoadTest.java`.
- A short description (a paragraph or two, but not more that that) of how did you implement `Vehicle.java, Highway.java`, and `ScenicRoad.java`, and what did you test them.. You can continue storing your written description into a file **writeup.txt**.

Final Check Point - What Should You Have Turned In

Each assignment will indicate exactly what to turn in paragraphs titled **What to Turn In**. This will typically include Java source files (that you change or create), and occasionally some text files. You will turn in your assignments by committing changes, and pushing those changes to your GitHub repository. To do so:

- Add and commit all changed (or newly-added) files.
- Push the committed changes to the repository so they are stored in GitHub, as well as updated locally.

You can commit and push changes as many times as you want while working on the assignment. That is a good way to store backup copies of your work in the CCIS GitHub repository.

When you have committed and pushed all of your changes, and you are done with the assignment, you should create a git tag in your repository named **Assignment1-final**, and push that tag to your repository too. Once you have committed and pushed that tag, your assignment has been submitted. **The staff will grade the files in your repository that are labeled with that tag, so be sure you remember to add/commit/push your files and the tag!**

Your TAs should be able to find the following in the src directory of CCIS GitHub:

- `Assignment1/RandomCourse.java` that prints out one of four random course names when its main method is executed.
- `Assignment1/Fibonacci.java` that passes the three tests in `Assignment1/test/FibonacciTest.java` (Note that you should not edit `Assignment1/test/FibonacciTest.java` to accomplish this task.)
- `Assignment1/ReverseNumber.java` and `Assignment/ReverseNumberTest.java`
- `Assignment1/Vehicle.java, Assignment1/Highway.java` and `Assignment1/ScenicRoad.java` that pass their respective JUnit tests, which you should with you include with your code as well.
- Short textual file, writeup.txt that describes you design and testing process for each of the problems.