Homework #3: Unit Testing & Continuous Integration

Due Tuesday, February 11th at 11:59 p.m.

In this assignment you will implement and submit unit tests for (1) a short `AvlTreeSet` class, and (2) the graph implementations you submitted for Homework 1. You will also configure your Git repository for the automated building and testing of your code. The learning goals for this assignment are:

- Write unit tests with JUnit with and without existing functional specifications.
- Automate builds and tests.
- Experience the benefits and limitations of code coverage metrics, and interpret the results of coverage metrics.
- Write high quality bugs reports and associate the bug fixes with the reports.
- Learn good testing practices and style.
- Generate meaningful, cohesive commits and reasonable commit messages.

This document continues by describing the tools you need to complete this homework, and then describes your actual tasks.

Installation & Setup

For this assignment, you will use three tools: Apache Ant, EclEmma, and Travis CI. We describe them each in turn.

1. **EclEmma** (http://www.eclemma.org) is a tool for visualizing and analyzing code coverage while testing Java applications in Eclipse. EclEmma monitors the execution of a Java application and highlights the lines of source code that have been fully executed at runtime (i.e., *fully covered*) in green. Lines of code that have been only partially executed or have not been executed are highlighted in yellow and red respectively. EclEmma also summarizes your code coverage as a percentage of the overall application, separately reporting instruction, branch, line, method, and type coverage. Please note that EclEmma will show you code coverage statistics for your unit tests as well; you can ignore that data.

   Installing EclEmma is very easy and can be done through the Eclipse Marketplace. See Piazza for more detailed instructions (including screenshots) on how to install the plugin.

2. **Apache Ant** (http://ant.apache.org) is a build and deployment software tool that uses XML to specify dependencies and build tasks. Ant is typically configured with a `build.xml` file that describes multiple tasks (usually called *targets*) that can be executed. These targets can be anything from building the project .jar file to running unit tests or static analysis tools on the code.

   Ant can be run in Eclipse or from the command line. To run from Eclipse, right-click `build.xml` and select Run As → Ant Build. To run from the command line (recommended), you will need to follow this guide for installing Ant (see Piazza for additional instructions). After installing a command line version of Ant you can `cd` into the directory where the `build.xml` file is located and execute Ant via this command:

   ```
   $ ant test
   ```

   The `test` argument specifies which target is run. The targets themselves can be named anything, but Travis CI (described below) runs the `test` target to build and test your project.

3. **Travis CI** (http://travis-ci.com) is a hosted continuous integration service that automates the building and testing of projects hosted on GitHub. Your repository has already been set up for Travis CI—note the `.travis.yml` file that has been added to the root of your repository. This configuration causes Travis CI will clone and build your project using `ant test` every time you push to GitHub. If your build fails (or if your unit tests fail), then Travis CI will send you mail, and can check the results on the Travis CI website.

Instructions

This assignment contains three distinct tasks, each of which are described below.

Configure Ant

Your repository contains two nearly empty `build.xml` files in the `homework/1` and `homework/3` directories. You must complete those configuration files so that `ant test` will compile and test your code using the unit tests you will write (in the next section). We recommend that you examine the sample `build.xml` file provided in recitation 4, and that you consult the Ant manual for more information. For this assignment we will not build your projects using Eclipse. Instead, we will compile and test your homework using the `build.xml` files that you complete.

We encourage, but do not require, that you investigate how you can automate additional tasks, such as generating javadoc and Ant files and running coverage and static-analysis tools such as FindBugs and Checkstyle with Ant. Be warned, though, that all your builds with `ant test` must complete within a time limit of one minute; we are enforcing this limitation so that each student can get a fair share of the build-and-test services we provide for this assignment.

We recommend that you complete the `build.xml` file for the `homework/1` *after* you set-up the directory structure for testing. See below for more instructions on how to accomplish that.

Unit test `AvlTreeSet.java`, reporting and fixing existing issues

AVL trees are self-balancing binary search trees that you can learn more about here: http://en.wikipedia.org/wiki/AVL_tree. We have provided an AVL tree implementation in `AvlTreeSet.java`, and its specification is included in the source code as Javadoc comments.

For this part of the assignment, you must:

1. Write unit tests to achieve 100% line coverage of the `AvlTreeSet` class.
2. Report an issue for each bug you find in `AvlTreeSet` on the GitHub issue tracker for your repository, which can be found at the following location:

   https://github.com/CMU-15-214/ANDREWID/issues

   Produce a high quality bug report each for each bug you find. Your report should include at least the following information: the context of the bug (i.e., where in the code the bug occurs), the error that occurs as a result of the bug, and (optionally) a suggested fix for the bug.
3. Fix each issue in an independent commit. Write meaningful commit messages and close the bug report. You can close an issue by adding '`fixes #xxx`' to your commit message where `#xxx` refers to the issue number or by referring to the commit from the message with which you close the bug. You can read more about fixing issues with commits here. If you do not close the issue with a commit, please close the issue manually (on the GitHub issue tracker page) and refer to the commit on the issue page.

Unit test your Homework #1 graph and algorithms

Write unit tests for your solution to Homework 1—all code submitted with Homework 1, including code provided by us. To accomplish this, you must:

1. Convert any tests you originally wrote for Homework 1 to JUnit tests. Organize your project in a meaningful way. The `public static void main(String[] args)` method is no longer needed.
2. Try to achieve 100% line coverage, but note that full coverage may not be achievable. If you cannot achieve 100% coverage, try to get as close as possible and explain with a short note in the uncovered area why you can't achieve 100% coverage. If necessary, you may modify your initial Homework 1 solution to help make your code more testable.
3. If you discover bugs in your original implementation, write bug reports and fix those bugs with a separate commit, as described for AVL trees above.

To complete this homework, you will need to write tests for both parts and automate those tests with Ant so that they successfully run on Travis CI. Commit all test files, build scripts, and libraries as necessary. Be sure to document all bugs in the issue tracker and fix all bugs, closing the corresponding issues with a specific commit.

Evaluation

Overall this homework is worth 100 points. To earn full credit you must:

- Correctly configure your `build.xml` files for Homework 3 and Homework 1 so that `ant test` compiles and tests your code correctly on Travis CI.
- Find all bugs in the provided `AvlTreeSet` implementation by writing correct unit tests on which the provided implementation fails.
- Document the bugs you find in the provided `AvlTreeSet` implementation with GitHub issues. Fix each bug with a separate commit with a meaningful commit message. Close the reported issue and associate it with the commit that fixes it.
- Create unit tests to achieve 100% line coverage for the provided `AvlTreeSet` implementation.
- Achieve high line coverage of your solution to Homework 1.
- Apply best practices to writing unit tests (i.e. write many small, independent tests instead of bunching them all into a single test case method, etc.).
- As always, make sure your code is readable and follows the Java naming conventions.

Additional hints:

- When specifications are available, write unit tests to ensure that the code you are testing meets the provided specification.
- Write your tests using JUnit 4 and measure line coverage using EclEmma. We will use these tools to evaluate your homework submission.
- You do not need to worry about the percentage of line coverage EclEmma reports for the `.java` files that contain your unit tests (EclEmma can be configured to exclude these packages, if desired). You also don't need to worry about code coverage in the `Main.java` file provided in Homework 1.
- Unit tests should be written using JUnit 4. JUnit is a unit testing framework for Java that is intended to make it easy to implement repeatable tests. Eclipse comes with JUnit installed by default. You can create a new unit test by clicking File → New → JUnit Test Case. You can find additional documentation and starter cookbooks for using JUnit at http://junit.sourceforge.net/.
- Do not modify the `.travis.yml` and `build.xml` files in the root of your repository. We enforce a timeout of 60 seconds for all builds to balance Travis CI capacity for all students.

We will grade your work approximately as follows:

- Build automation with Ant and Travis CI: 20 points
- Adequate code coverage in testing the `AvlTreeSet`: 25 points
- Adequate code coverage in testing Homework 1: 25 points
- Documentation of bugs via the GitHub issue tracker and bug fixing: 20 points
- Compliance with unit testing best practices: 10 points