Computer Science 241

Lab 3 (10 points)
Due Sunday, April 29th, 2018 at 10:00 PM

Read all of the instructions. Late work will not be accepted.

Overview

Testing is an important part of software development; generally, testing is done to assess whether a software product will properly serve its intended purpose. Many different parties are involved in testing over the lifetime of a software product, and there exist many different methods of software testing. This lab focuses on *unit testing*, which aims to assess the correctness/usefulness of individual components of a larger program. Sometimes the unit to be tested in a single class, but you can also unit test individual methods; we will use the latter for this lab. The correctness of each individual unit in a larger program can be thought of as a necessary, but not sufficient, condition for the program to work. Testing individual units can also make it much easier to locate and fix bugs, a topic further explored in a later lab.

Lab Description

You will be implementing and then testing an implementation of insertion sort. I expect that the testing will require a more time and effort than coding up the sort itself. This isn't uncommon for software testing. *Remember to* `git add`, `git commit` *and* `git push` *regularly while developing your code; e.g. one or more times per method you implement.*

InsertionSort Class

Under a `lab3` subdirectory of your repository, create a `InsertionSort.java` file that implements a public class `InsertionSort`. The class should have a `public static` method named `insertionSort` that takes an array of integers as input and returns an array of integers (you may add `private` helper methods as needed). The method should implement the insertion sort algorithm as presented in class. It is this `insertionSort` method that you will be unit testing.

TestInsertionSort Class

Under the same `lab3` subdirectory of your repository, create a `TestInsertionSort.java` file that implements a public class `TestInsertionSort`. This class will contain several methods for testing your `insertionSort` method, including:

- A `private static boolean isSorted` method that takes as input an array of `int`s.
  - return `true` if the array is sorted in ascending order and return `false` otherwise
  - Hint: this check can be done $O(N)$

- A `private static boolean sameElements` method that takes as input two `int` arrays
  - return `true` if both arrays have same counts of elements, and return `false` otherwise
  - The elements need not be in the same order: `2 3 2 1` and `1 2 2 3` have the same counts of elements
  - Hint: consider using a `java.util.HashMap<Integer,Integer>` to keep track of how many times each value appears
- A `private static void testFromConsole` method that takes no arguments and repeats the following five steps until the user enters `ctrl+c` to kill the process:
  1. Read a line's worth of `int`s from standard in and store this as an `int` array
  2. Sort the `int` array using your `insertionSort` method
  3. Feed output of `insertionSort` into `isSorted`
  4. Feed the original (pre-sort) `int` array and the output of insertion sort into `sameElements`
  5. If both `isSorted` and `sameElements` return true, print `Passed Test` to standard out; else print `FAILED Test`
  - Note that `testFromConsole` allows manual testing, which is one way to unit test
- A `private static void shuffleTest` method that takes as input an `int` N, and does the following five steps (one time only, then returns):
  1. Generates an integer array of length N, containing the elements $0, 1, \ldots, N-1$ in ascending order
  2. Randomly shuffles/permutes the array (save a copy of the unshuffled array)
     Hint: you can use a built-in to shuffle (e.g. `java.util.Collections.shuffle`)
  3. Uses `insertionSort` to sort the shuffled array
  4. Compare the output of `insertionSort` to the original ascending array
     Hint: You should do an array comparison; e.g. `Arrays.equals`
  5. If the sorted array equals the original array, print `Passed Test`; else print `FAILED Test`
  - Note that `shuffleTest` is an automatic test, which is another way to unit test
- A `public static void main` method that behaves as follows:
  - If there are no commandline arguments, it runs `testFromConsole`
  - If there are one or more commandline arguments, it parses them as `int` values and runs `shuffleTest` once per argument, using the argument as N. For example, if you run your program as

    ```
    java TestInsertionSort 1 10 1000
    ```

it would call `shuffleTest` three times: once with $N = 1$, once with $N = 10$ and once with $N = 1000$.

Note: if during your testing you encounter any FAILED tests, make sure to fix your `insertionSort` *and/or test methods to resolve the issue.*

Submission

The `master` branch of the `origin` repository (i.e. the one I made for you in the `hutchteaching` organization) should contain the following files:

- `lab3/InsertionSort.java`
- `lab3/TestInsertionSort.java`

You can confirm that your code is properly submitted by checking your github repo URL:

`https://github.com/hutchteaching/201820_csci241_username`

Grading

At the deadline a script will automatically clone your repository. Points will be deducted for any problems in your submission, including:

- Missing or incorrectly named files or directories
- Code that does not compile
- Code that generates run-time exceptions
- Missing or incorrectly implemented `TestInsertionSort`
- Missing or incorrectly implemented `InsertionSort`
- Other assorted failure to follow Lab 3 instructions
- Poorly coding style (e.g. inconsistent indentation, silly variable names)