Programming Assignment #2: GenericBST

COP 3503, Spring 2017

Due: Sunday, February 5, *before* 11:59 PM

Abstract

In this assignment, you will gain experience working with generics in Java. In particular, you will extend my binary search tree (BST) code from a data structure that just holds integers to a powerful container that can hold any kind of Comparable object under the sun.

A tangential benefit of this program is that you get to spend some time looking through some fairly straight-forward and well-structured BST code. It is always good to revisit how common data structures can be implemented as you ramp up your skills in different programming languages. You will also gain experience modifying someone else's code (something you will be doing a lot if you become a software developer) and adding comments to code.

Deliverables

GenericBST.java

(Note: Capitalization of your filename matters!)

1. Problem Statement

Modify the attached `BST.java` source file as follows:

1. Change the class name to `GenericBST` and the file name to `GenericBST.java`.

2. Modify the source code so that the class is generic (i.e., so that the binary search trees will be able to hold any type of data) with the restriction that only classes that implement `Comparable` may be stored in the BSTs.

3. The comments in `BST.java` are practically non-existent. After you have read and absorbed the program's structure and functionality, add comments for the entire program. In general, try to keep comments brief, and prefer comments that explain *how* or *why* rather than just labeling *what* a block of code is. For example, the following *what*-style comment is not useful:

   ```
   // Main method
   public static void main(String [] args) { ... }
   ```

   Be sure to include your name and NID in a header comment.

4. Write a `public static double difficultyRating()` method that returns a double on the range of 1.0 (ridiculously easy) through 5.0 (insanely difficult) indicating how difficult you found this assignment to be.

5. Write a `public static double hoursSpent()` method that returns an estimate (greater than zero) of the number of hours you spent working on this assignment.

To make this class generic, you will have to change the return types of some of the methods I have implemented. However, please do not change static methods to non-static methods, or private methods to public methods, and vice-versa.

2. Compilation Requirements: No Compile-Time Warnings

Your source code must not produce any warnings on Eustis when compiled.

Please note: Some compilers don't produce -Xlint warnings! It's important that you know for sure that you're using a compiler that generates such warnings. There are two ways to be sure of that:

1. Use Eustis to compile your program. It will give -Xlint warnings.

2. For your convenience, I have included a file called `GenericsWarning.java`, which should produce a warning when you try to compile it. If your compiler does *not* produce a warning when you compile `GenericsWarning.java` (or, perhaps, while editing it in your IDE), please seek out a compiler/IDE that does.

Please do not give your code to classmates and ask them to check whether their compilers generate compile-time warnings. Remember, sharing code in this course is out of bounds for assignments.

3. Compiling and Testing GenericBST on Eustis

Recall that your code must compile, run, and produce precisely the correct output on Eustis in order to receive full credit. Here's how to make that happen:

1. To compile your program with the test case I included with this assignment:

```
javac GenericBST.java TestCase01.java
```

2. To run this test case and redirect your output to a text file:

```
java TestCase01 > myoutput01.txt
```

3. To compare your program's output against the sample output file I've provided for this test case:

```
diff myoutput01.txt TestCase01-output.txt
```

If the contents of `myoutput01.txt` and `TestCase01-output.txt` are exactly the same, `diff` won't print anything to the screen. It will just look like this:

```
seansz@eustis:~$ diff myoutput01.txt TestCase01-output.txt
seansz@eustis:~$ _
```

Otherwise, if the files differ, `diff` will spit out some information about the lines that aren't the same.

4. Test Cases and the test-all.sh Script

I've included a script, `test-all.sh` that will compile and run the included test case for you. You can run it on Eustis by placing it in a directory with `GenericBST.java` and all the test case files and typing:

```
bash test-all.sh
```

5. Miscellaneous Requirements

Please be sure to submit your `.java` file, not a `.class` file (and certainly not a `.doc` or `.pdf` file). Your best bet is to submit your program in advance of the deadline, then download the source code from Webcourses, re-compile, and re-test your code in order to ensure that you uploaded the correct version of your source code.

Special restriction! Your code must not produce any compiler warnings (especially -Xlint warnings), and you cannot use the `@SuppressWarnings` annotation to suppress them.

Important! Programs that do not compile will receive zero credit. When testing your code, you should ensure that you place `GenericBST.java` alone in a directory with the test case files (`TestCase01.java` and `TestCase01-output.txt`), and no other files. That will help ensure that your `GenericBST.java` is not relying on external support classes that you've written in separate `.java` files but won't be including with your program submission.

Important! You might want to remove `main()` and then double check that your program compiles without it before submitting. Including a `main()` method can cause compilation issues if it includes references to home-brewed classes that you are not submitting with the assignment. Please remove.

Important! Your program should not print anything to the screen. Extraneous output is disruptive to the TAs' grading process and will result in severe point deductions. The only output should come from the pre-written tree traversal methods. Please do not make any changes to the print statements in the tree traversal methods.

Important! Please do not create a java `package`. Articulating a `package` in your source code could prevent it from compiling with our test cases, resulting in severe point deductions.

Important! Name your source file, class(es), and method(s) correctly. Minor errors in spelling and/or capitalization could be hugely disruptive to the grading process and may result in severe point deductions. Similarly, failing to make any of the three required methods, or failing to make them `public` and `static`, may cause test case failure. Please double check your work!

Test your code thoroughly. Please be sure to create your own test cases and thoroughly test your code. You're welcome to share test cases with each other, as long as your test cases don't include any solution code for the assignment itself.

6. Grading Criteria

The *tentative* scoring breakdown (not set in stone) for this programming assignment is:

40%	program passes test cases (in which I'll throw some home-brewed comparable objects into a `GenericBST` object and see whether the trees are properly constructed)
40%	program compiles without warnings, method signatures are correct, and generics with `Comparable` are properly implemented
5%	`difficultyRating()` returns a double in the specified range
5%	`hoursSpent()` returns a double in the specified range
5%	adequate comments and whitespace; source code includes student name
5%	source file is named correctly (`GenericBST.java`); capitalization counts

Start early! Work hard! Ask questions! Good luck!