Programming Assignment #2: GenericBST

COP 3503, Fall 2013

Due: Sunday, September 22, 11:59 PM via Webcourses@UCF

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

1. Problem Statement

Modify the attached BST.java source file as follows:

1. Change the class and file names to GenericBST.

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

4. Write a `public static double difficultyRating()` method that returns a double on the range of **1.0** (ridiculously easy) to **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.

Please note: You should not assume `compareTo()` returns -1, 0, and 1. In place of -1, `compareTo()` methods may return *any* negative integer. In place of +1, `compareTo()` methods may return any *positive* integer. (Zero, of course, still indicates equality.) I have elaborated on this in Webcourses.

2. What to do with main()

You can do whatever you want with `main()`. You don't have to remove it or comment it out. You can just leave it in there. I will write an AutoGrader class that uses your GenericBST class; your `main()` method will not be called.

3. Compilation Requirements: No Compile-Time Warnings

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

4. Grading Criteria and Miscellaneous Requirements

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
10%	`difficultyRating()` and `hoursSpent()` return doubles in the specified ranges
5%	adequate comments and whitespace
5%	source file is named correctly (`GenericBST.java`)

Programs that do not compile will receive zero credit. 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.

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.

Please do not create a java `package`. Articulating a `package` in your source code could be disruptive to the grading process and will result in severe point deductions.

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. Please double check your work!

Test your code thoroughly. Please be sure to create your own test cases and thoroughly test your code.

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