Joanna Klukowska
joannakl@cs.nyu.edu

# Project 6: Implementing a Binary Search Tree

Due date: Decemeber 8, 11:55PM EST.

**You may discuss any of the assignments with your classmates and tutors (or anyone else) but all work for all assignments must be entirely your own . Any sharing or copying of assignments will be considered cheating (this includes posting of partial or complete solutions on Piazza, GitHub or any other public forum). If you get significant help from anyone, you should acknowledge it in your submission (and your grade will be proportional to the part that you completed on your own). You are responsible for every line in your program: you need to know what it does and why. You should not use any data structures and features of Java that have not been covered in class (or the prerequisite class). If you have doubts whether or not you are allowed to use certain structures, just ask your instructor.**

---

In this project you will implement a binary search tree class based on a given specification. The specification is similar to the one for `TreeSet` class provided by Java libraries, but your implementation will be very different.

## Objectives

The goal of this programming project is for you to master (or at least get practice on) the following tasks:

- implementing and working with a binary search tree structure
- testing your own implementaiton
- using and implementing iterators
- implementing generic classes
- implementing existing interfaces (`Iterable`, `Iterator`)
- (optionally) implementing a balanced binary search tree (AVL)

Make sure to ask questions during recitations, in class and on Piazza.

**Start early!** This project may not seem like much coding, but testing and debugging always takes time.

**For the purpose of grading, your project should be in the package called `project6`. This means that each of your submitted source code files should start with a line:**
**`package project6;`**
**Keep in mind that spelling and capitalization are important! The package declaration line has to be the first line in your file!**

## `BST<E>` Class

The specification for this class is provided at its javadoc page.

You can use the source code that we wrote in class, but keep in mind that the class that you are implementing is generic and we worked with a concrete class with integers stored in the nodes.

### Nodes

The program should provide and use a nested class[1] that provides nodes for your list. The details of the implementation of that class are up to you, but this class should be private and static:

```
private static class Node <E>
```

**HINT:** to improve the performance of your BST algorithms, it may be useful to keep additiional data fields in the nodes, i.e., more than just `data`, `left` and `right`. Those design decisions are up to you. But you should explain in comments for this class, why you have additional data fields if you chose to do so.

[1] To learn more about nested and inner classes see: https://docs.oracle.com/javase/tutorial/java/javaOO/nested.html

Joanna Klukowska
joannakl@cs.nyu.edu

### Iterator

The **`BST<E>`** class implements **`Iterable<E>`** interface. This means that its **`iterator()`** method needs to return an instance of a class that implements the **`Iterator<E>`** interface. The **`iterator()`** method should return an iterator instance that accesses the values in the tree according to the inorder traversal of the binary search tree. The two additional methods **`preorderIterator()`** and **`postOrederIterator()`** need to return iterators that access the values in the tree according to the preorder and postorder traversals, respectively.

The details of the implementation are up to you and you are free to implement more than one internal private iterator class. The **`remove`** method in the **`Iterator<E>`** interface is optional and you do not need to provide the actual remove functionality. (This means that the method has to exist, but instead of performing its function, it throws an instance of `UnsopportedOperationException`.)

## Extra Credit AVL Implementation - 20 points

Once you are done with your **`BST<E>`** implementation, you can try to convert it to a class that implements an AVL self balancing tree. This should require changes to only a few of the function that you implement for the **`BST<E>`** class.

The **`AVL<E>`** class should follow exactly the same specification as the **`BST<E>`** class.

There will be a separate submission link for the **`AVL<E>`** class. You will need to submit your **`BST<E>`** class to one link and your **`AVL<E>`** class to another link. (You cannot submit the AVL tree implementation to both since some of the test for BST will fail, if you do so.)

## Programming Rules

You should follow the rules outlined in the document *Code conventions* posted on the course website at https://cs.nyu.edu/~joannakl/cs102_f18/notes/CodeConventions.pdf.

You may use any exception-related classes.

**For the binary search tree implementation you should use the textbook, lecture material and source code of Java built-in classes as a guide. You are free to look at the `TreeSet` class implemented in Java, but be warned that that class implements a red-black balanced binary search tree.**
**As usual, you should give credit to any sources you are using.**

## Working on This Assignment

You should start right away!

You should modularize your design so that you can test it regularly. Make sure that at all times you have a working program. You can implement methods that perform one task at a time. This way, if you run out of time, at least parts of your program will be functioning properly.

You should make sure that you are testing the program on much smaller data set for which you can determine the correct output manually. You can create a test input file that contains only a few rows.

You should make sure that your program's results are consistent with what is described in this specification by running the program on carefully designed test inputs and examining the outputs produced to make sure they are correct. The goal in doing this is to try to find the mistakes you have most likely made in your code.

Each class that you submit will be tested by itself without the context of other classes that you are implementing for this assignment. This means that you need to make sure that your methods can perform their tasks correctly even if they are executed in situations that would not arise in the context of this specific program.

**You should backup your code after each time you spend some time working on it. Save it to a flash drive, email it to yourself, upload it to your Google drive, do anything that gives you a second (or maybe third copy). Computers tend to break just a few days or even a few hours before the due dates - make sure that you have working code if that happens.**

Joanna Klukowska
joannakl@cs.nyu.edu

# Grading

If your program does not compile or if it crashes (almost) every time it is run, you will get a zero on the assignment. Make sure that you are submitting functioning code, even if it is not a complete implementation.

If the program does not adhere to the specification, the grade will be low and will depend on how easy it is to figure out what the program is doing.

**40 points** class correctness: correct behavior of methods of the required class

**20 points** design and the implementation

**20 points** efficiency of the implementation (the expected performance is mentioned in the specification of each of the methods)

**20 points** proper documentation, program style and format of submission

**20 points** extra credit AVL tree implementation

# How and What to Submit

**For the purpose of grading, your project should be in the package called `project6`. This means that each of your submitted source code files should start with a line:**

```
package project6;
```

Your should submit all your source code files (the ones with .java extensions only) in a single **zip** file to Gradescope. (You can also submit the single file that you are working on directly, without packaging it in a zip file.)

You can produce a zip file directly from Eclipse:

- right click on the name of the package (inside the `src` folder) and select Export...
- under General pick Archive File and click Next
- in the window that opens select appropriate files and settings:
  - in the right pane pick ONLY the files that are actually part of the project, but make sure that you select all files that are needed
  - in the left pane, make sure that no other directories are selected
  - click Browse and navigate to a location that you can easily find on your system (Desktop or folder with the course materials or ...)
  - in Options select ”Save in zip format”, ”Compress the contents of the file” and ”Create only selected directories”
- click Finish