CS146: Spring 2023
Homework 3: Trees
Due Friday, March 3, at 11:59PM
100 points

For this homework you will implement some methods for Red-Black Trees. This homework will come with a 15 minute grading interview (see Canvas for signup details). The interview will take place via Zoom with me (or the grader) and you will have to share your screen with your code. Questions will include details around the homework assignment and the material covered in class around red-black trees and b-trees (e.g., I may ask you to describe how you would insert a number into a b-tree).

To get started, import the starter file, RBTree.java, into the trees package you create in a new Java Project. You CANNOT modify the inner Node class that contains information about the nodes of the binary trees. Please do not change any of the method signatures in the RBTree class, but you are free to add additional helper methods. This is your most complex assignment so far. It is very important that you start early and test your code thoroughly. I also recommend familiarizing yourself with the debugger in Eclipse. It can be very helpful to see the structure of the tree.

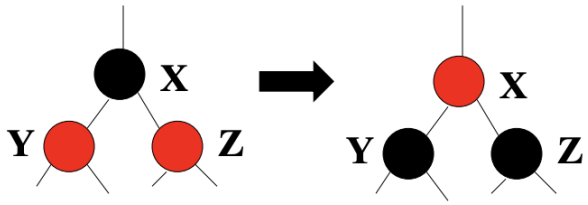
Part 1: Top Down Insertion for Red-Black Trees (60 points)

```
public boolean insert(Integer i)
```

This method should use a top down insertion strategy (see below) to insert a Node with data equal to i into the Red-Black Tree **provided that a Node with data equal to i does not already exist in the Red-Black Tree** (i.e., no duplicate data). If the node is successfully inserted, return true, otherwise return false (tried to insert a duplicate element).

For the first part of the assignment, you will implement a top-down insertion for a red-black tree. The general outline of the algorithm is described below (adapted from https://www.rose-hulman.edu/class/cs/csse230/201420/Slides/15-RedBlackTrees.pdf). Basically the idea is to do the recoloring and rotations on the way down instead of after you insert the node. Node that the node to be inserted is always red, you may need to do one final rotation after inserting the node if its parent is red. Note that in this version of the red-black tree, we do not have parent pointers in our Node class. **You cannot add a parent pointer to the node class or do additional tree traversals to find parent nodes**. You will need to use a Stack to keep track of the ancestors in order to perform the correct rotations.

This is the situation you need to handle on your way down the tree

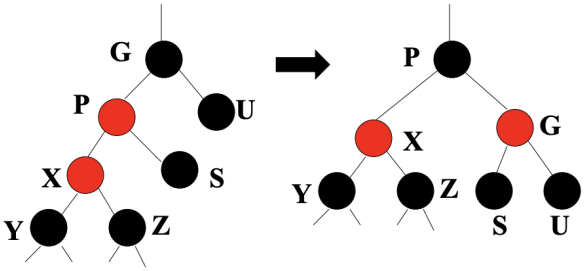


Situation: A black node with two red children.
Action: - Recolor the node red and the children black.
- If the parent is red, perform rotations, otherwise continue down the tree

If the color flip produces a red node with a red child, then perform a single or double rotation depending on the following…

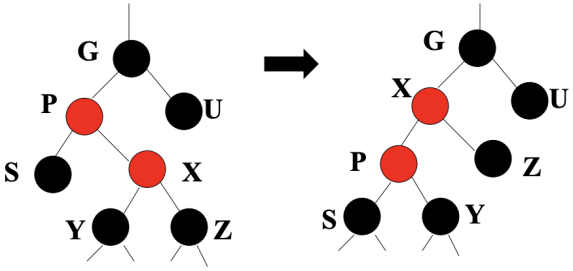
- If the two red nodes are both left children or both right children, perform a single rotation (see example below)
- Otherwise, perform a double rotation (see example below)

Single Rotation on Left Child

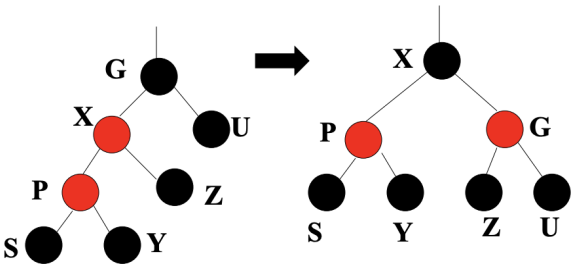


- The rotation is done on X's grandparent, G.
- The colors of P and G are flipped.

Double Rotation on Left Child



- Again, the rotation is done on X's grandparent, G.



- Recolor X and G

Testing Your Insert Method (10 points)

I have given you a very limited set of JUnit tests to help determine if your implementation is correct. You need to add at least 5 more JUnit tests for the insert method that test typical and edge cases. You get one point for creating the test case and one point for passing the test case. You are welcome (and encouraged) to create more than five JUnit tests, but as long as you create and pass five JUnit tests, you will receive full credit for this part of the assignment.

Part 2: Other methods on Red-Black Trees (10 points)

`public int height()`

This method returns the height of the red-black tree. The height is defined as the longest path from the root to a leaf node. Note that the root has a height of 1.

`public int blackHeight()`

This method returns the height of the black height of red-black tree. The height black is defined as the number of black nodes from a given node to a leaf node (the NIL node). Note that we do not count the current node in the black height, but we do count the NIL leaf node (e.g. the black height of the root in the red-black tree below is 2). This method can be helpful when testing your insert method to make sure that you do not violate the fifth property of red black trees.

Testing

While you won't get points for writing test cases for these methods, it is very important that you test your code before you submit it. If your code has compilation errors for these methods, you will receive an automatic 5 points deduction).

Part 3: RBTree Iterators (30 points)

You will implement three different iterators for your Red-Black Tree: a pre-order iterator, an in-order iterator, and a post-order iterator. You are free to add whatever instance variables and helper methods you need in addition to implementing the required constructor, next() and hasNext() methods [Note that other two methods have default behavior so we don't see explicitly implement them]. You are not required to implement those methods (remove and forEachRemaining). For more information about iterators see the Java Documentation.

Testing

While you won't get points for writing test cases for these methods, it is very important that you test your code before you submit it. If your code has compilation errors for these

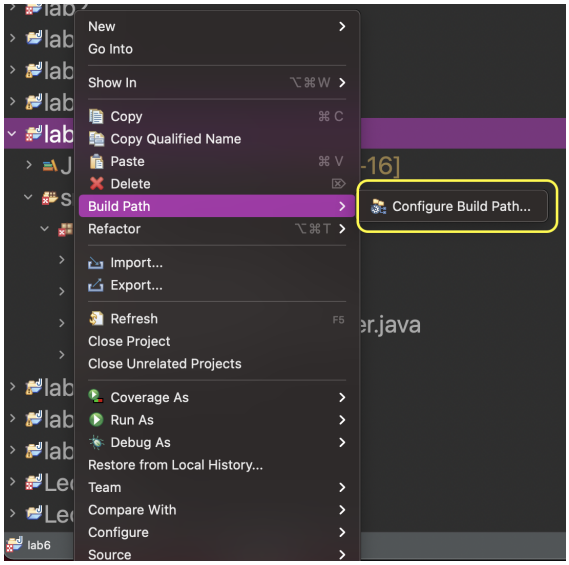
methods, you will receive an automatic 5 points deduction for each iterator that has a compilation error.).

Submitting

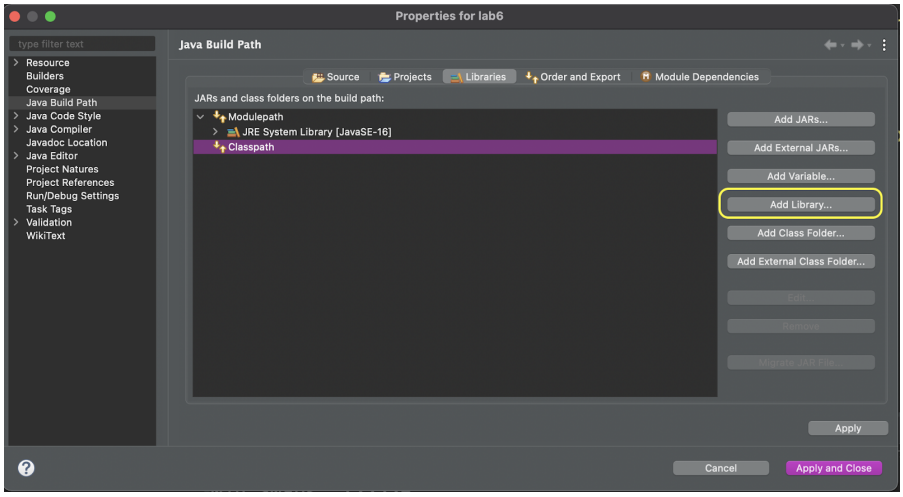
Export your project and upload it to Canvas. You can export your project as either a jar or a zip file. Make sure that the file you submit contains all your .java files in the package. If you upload a file that is missing any of the java source files you will get an automatic deduction of 10 points.

JUnit Setup

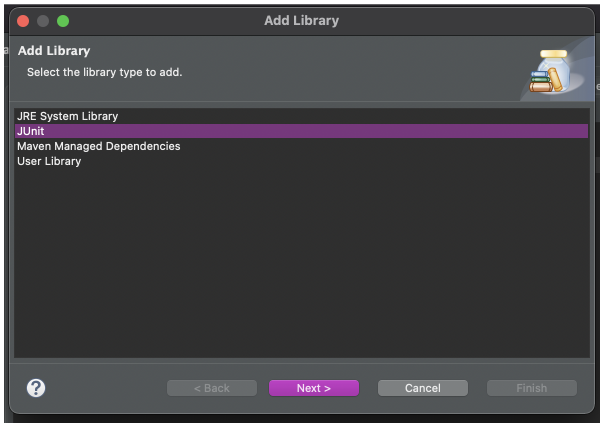
These instructions are for the Eclipse IDE. While you are welcome to use any IDE or programming environment, I will only provide instructions for Eclipse. To add the JUnit library to your build path, right-click on your homework project, look for Build Path and click on the Configure Build Path option.



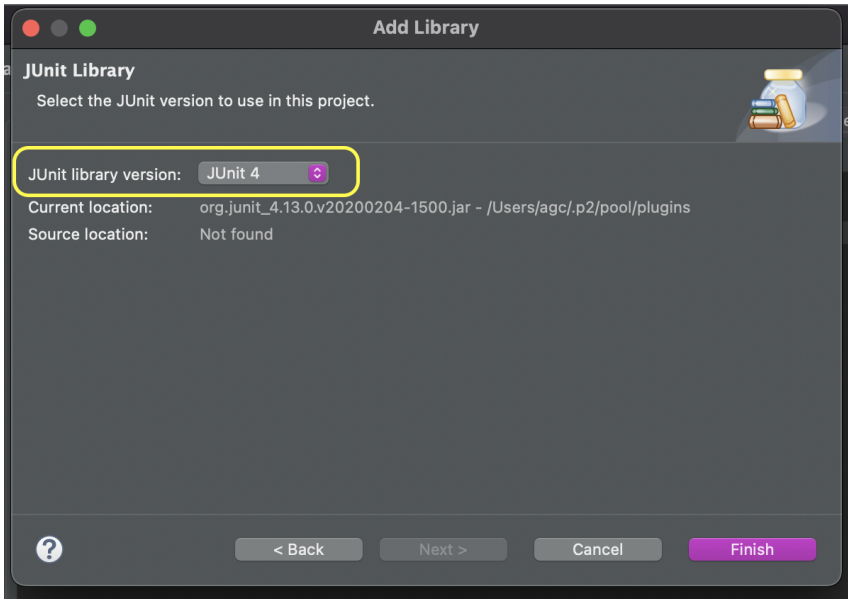
Highlight Classpath and click Add Library



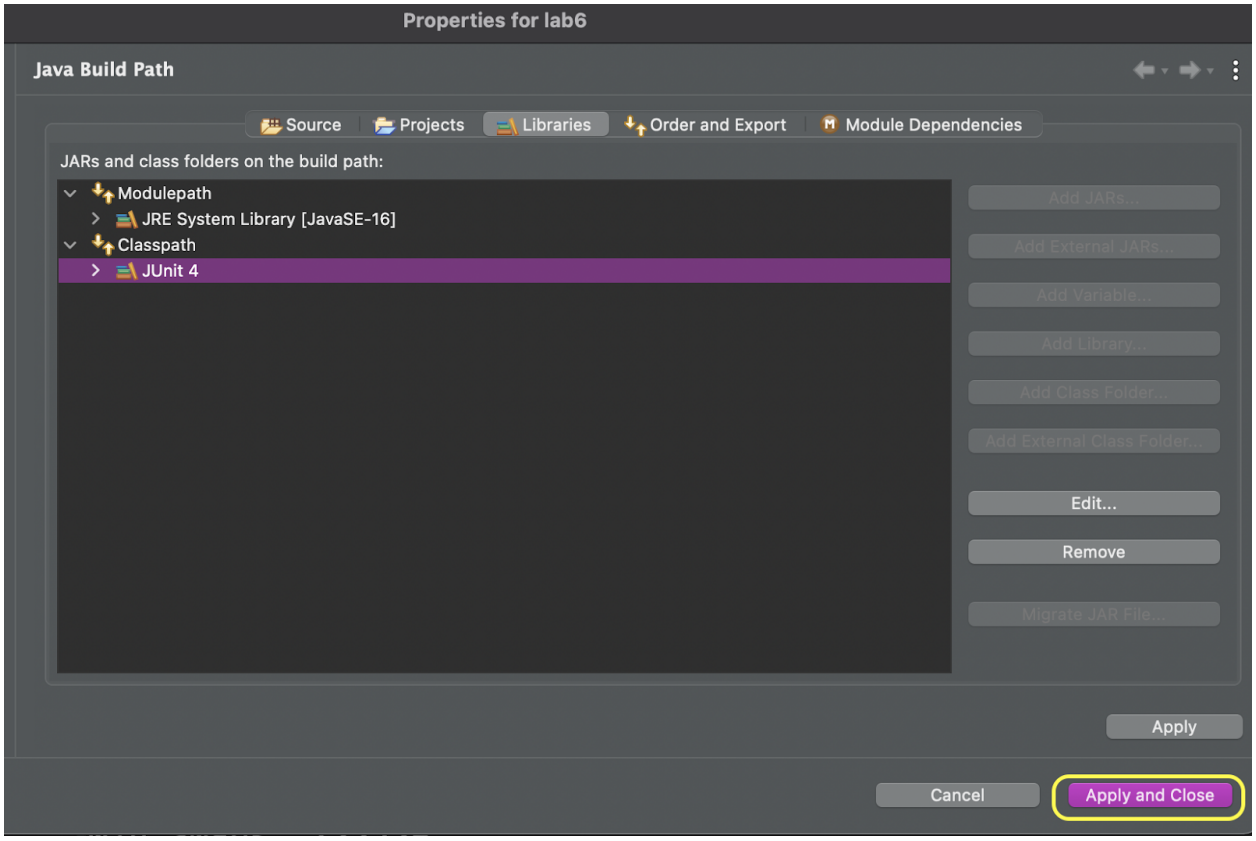
Select JUnit and click Next



Make sure you select JUnit 4. Then click Finish.



Your build path should look something like this. Click Apply and Close.



That should have gotten rid of any compilation errors you had when you originally copied over the files.