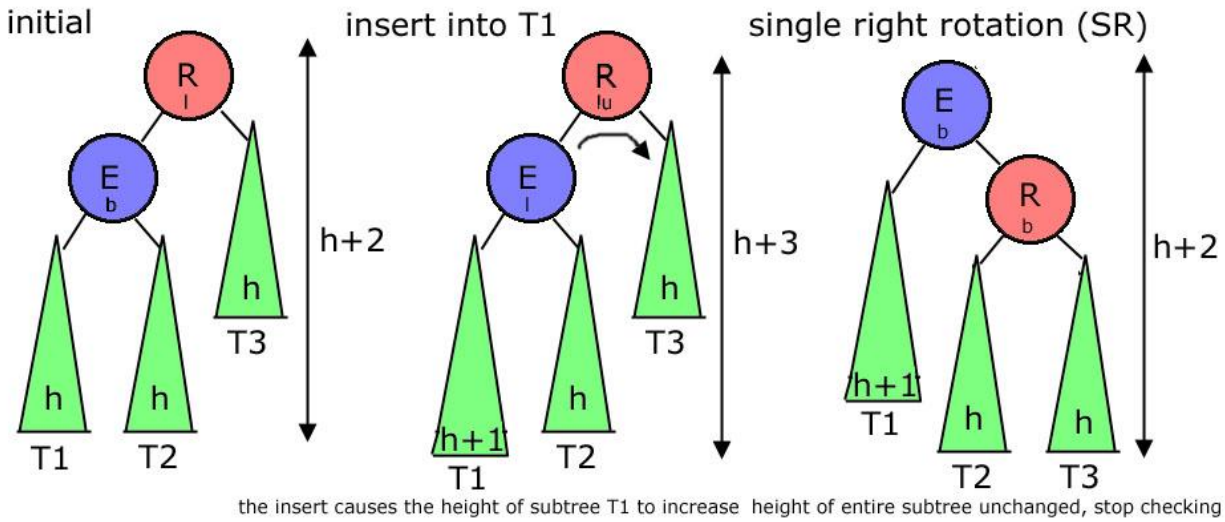
Laboratory 10: Method Cohesion and Implementation Inheritance Coupling

Java API
KeyedItem, CD, Song Documentation
FileIO Documentation

An AVLTree "is-a" BinarySearchTree that is also balanced. The given AVLTree implementation has several elegance problems. You will improve the AVLTree implementation.



Lab:

- Method Cohesion
- Method Decomposition
- Implementation Inheritance Coupling

Part I: Method Cohesion

- BinarySearchTree.java
- AVLTree.java
- AVLTreeNode.java

Add assert statements in AVLTree methods to check for correct balancing of AVLTree. Your assert statements will call the isBalanced() method that you wrote in Lab 09. Using your driver from the previous lab, verify that the AVLTree is balanced after any operation that modifies the tree has completed. That is, the assert statements should never indicate a problem. Remember to use the -ea option to activate the assert statements.

Identify the two methods in AVLTree which are not cohesive (this should be obvious). Correct this by writing new methods. This is "Replace Parameter with Explicit Methods" refactoring. **Use the fact that adding a node on a given subtree's left can only generate a SR or a DLR, and vice versa for a node on a given subtree's right**.

Part II: More Method Decomposition

Use the short methods that you created in the previous lab to simplify the code in AVLTree. By overriding the appropriate short methods in BinarySearchTree (but still calling super as appropriate), you should be able

to **eliminate the insertItem method in AVLTree**. It is possible to also eliminate deleteItem, but this is not required.

You will find that there is a lot of simplification/cleanup possible in the long methods that maintain the balance of the tree. This is "Extract Method" refactoring. Create new methods for each type of rotation to make the overall code responsible for rotations during an **insert only** as readable as possible. Also, you should be able to eliminate many balance factor assignments and conditional tests. **I will be looking at these methods carefully**.

Using your driver from the previous lab, apply unit testing as you work. That is, thoroughly test your modified AVLTree class after each modification. For example, after working on the single left rotation method, create a test case that will test this rotation. You are likely to see the advantages of using **assert** to check class invariants at this point.

Part III: Design Elegance

Think about your redesigned BinarySearchTree and AVLTree hierarchy. Write a one-half to one-page (double-spaced) paper discussing how your new design is more or less elegant than the original design. Discuss implementation inheritance coupling, the Liskov Substitution Principle, method cohesion, and whether you could and/or should have used referencing (has-a) instead of inheritance (is-a). Make sure to list your references (no specific format required).

You are required to choose one partner for this lab from your lab section. Only one submission per team is necessary, but please make sure to include both names at the top of your source code, as well as in the comments section when submitting the lab, so both people can get credit.

The Lab is due the following Sunday before midnight.