COSC 310 Program – AVL Tree Name Database, v1.0b

Goal: Practice data structures by implementing an AVL Tree and using it to create a name lookup.

Description: Congratulations! You have been hired by the *Steve K Ninja Database Corporation* to create the first version of their new product, *The Amazo Name Storage Program Shadow Magic 2000* (name subject to change)™. This product will be eventually be used to eradicate world hunger or that's what they tell you anyway, but luckily, the first version of this product is *far more modest*. You are asked to create a simple name database using an AVL Tree. (No one knows how this will eradicate world hunger, but this is still the prototyping phase after all…)

> The textbook provides an incomplete AVL implementation. Implement the code that they provide and complete it. Remember that an AVL tree is a BST with additional balance requirements. You will need to implement the AVLNode class and the balancing methods as discussed in class. Once you have a working tree, you can use it to implement a small database application. The database will read an input file and insert all of the data, allowing you to run the program as described below. *You will find descriptions of the various trees along with the code that you need to begin in chapter 4*.

To begin, implement all of the code from the textbook. This will give you most of, but not all of an AVL Tree—you will need to implement the single and double rotation methods on your own.

> At this point, you should have a working AVL Tree, which you can use for the application part of this assignment. Your database should be able to read the provided text file, inserting each record into your AVL Tree, sorting records by last name, then first name. *Do not allow multiple records with the same two names – instead printing an error message for that case. Then you should be able to search for users by name. If they give you "Kennedy", then print out the first Kennedy record, but if they give you a specific name like "Jimbarb Kennedy", then print out that record if it exists.

> Each line in the input file represents a record, except the lines that begin with #, which are comments that can be ignored. Your records contain the ID, First and Last Name, and Age of various randomly generated people. Read the file and then insert each name into your tree. You should accept user input (command line is okay), implementing the following commands for the user to enter when prompted. The commands in brackets [like this] are options for the user.

- find [name] – Searches the tree for the first person with that last name and prints out their record if present.
- find [fname] [lname] – Searches the tree for the person specified and prints their record if present.
- add [id] [fname] [lname] [age] – Inserts a new record with this info or generate an error message if it is invalid.
- count – Print the number of names in the database.
- who? – Print your name. (i.e. the creator of the code)
- ? – Print this list of commands for the user.
- exit – Quit the program.

> Your main application code will implement the provided interfaces (which conveniently ask you to create methods for each of these instructions). Your records will implement Record. Include these files in your project, but do not modify the files provided. Name your main class LnameFnameNames (where

Lname and Fname are your names), and name your record class FLRecord (where F and L are your initials). Create a user-friendly program with professional code quality. Document your code thoroughly (JavaDocs recommended) and save your code regularly.

Finally, make sure that you complete the Post Mortem for your Ninja Project Manager. A form has been included with this project.

Tips:

1. It can be hard to know whether your data structure is working. If you are familiar with the debugger in your IDE, it can help you, but you may also want to make clever use of System output to help you. Remember that an in-order traversal of your tree should always print a sorted list. That fact can help you validate operations on your tree.
2. Start small: Only use a few names at first for input and do features one at a time.
3. Think in modules: Keep separate features separate in code and you'll have an easier time creating them and fixing them.
4. Create a class file representing the records. You may want to implement Comparable.
5. Reach out to me with questions.

Please contact your Steve K Ninja Database Corporation service representative with questions or concerns. A checklist has been provided below for your convenience.

Check	Item	Value
X	Read specs carefully	0
	AVL Tree, textbook version	2
	Single rotation	2
	Double rotation	2
	Record	1
	Names Application	2
	Documentation	1
	Post Mortem	0
	Polish / Professionalism	+X
	Typos or missing deliverables	-Y++
	Code errors	-Z%