Assignment 3 (due 2pm Monday week 7)

Binary Trees: Shop Items

New this week: *Binary trees, Files*

Deadline for submission: Monday 2nd March, 2pm

Hand-in Method: Submit on MyDundee. Note that this is an individual assignment (not groupwork!).

Date feedback will be received by: Latest date to receive feedback will be Monday, 23rd March.

Penalty for late submission: One grade point per day late (meaning if a submission is one day late and marked as a C2 it will receive a C3 grade) A day is defined as each 24-hour period following the submission deadline including weekends and holidays. Assignments submitted more than 5 days after the agreed deadline will receive a zero mark (AB).

Percentage of Module: This assignment is worth 10% of this module for AC12001 and 7% for AC22007.

Assessment Details

Everyone should attempt the first four requirements below (out of 80%); you should only attempt the final requirement(s) if you feel confident that you have finished all of the others and wish to try some more. Remember, your report must state which requirements were met and what problems, if any, you encountered.

Aim

The aim of this lab is to learn how to design and code a program that uses a binary search tree data structure without using the Java collection classes, i.e. you must code your own binary search tree from first principles and not use a pre-built library tree class. You will also gain further experience of using recursion. For this task you will store shop item information as nodes in a binary tree.

Requirements

1. Allow the user to enter details for a shop item (item ID number, name and cost). The program should add the item as a note into a binary tree.
2. Allow the user to request that the whole tree be printed in numerical order of Item ID.
3. Allow the user to enter an Item ID to look for in the tree and show the name and cost for that shop item (or display a message if the item does not appear in the tree).
4. Calculate the total cost of all the items in your Tree.
5. Allow the user to request that a particular item is removed from the tree (or display a message if the item does not appear in the tree).
6. *[Optional]* Write out your tree to a text file, and then read it in again at the start of the program
7. *[Optional]* Implement other forms of traversal for the tree, e.g. do a preorder and postorder traversal to display the tree contents.
8. *[Optional]* Extend your program to keep track of stock levels of all Items by adding a field in the TreeNode to count the number of Items in stock. When you add an item that exists it will then increment the count of that Item. If you remove an Item, it will reduce the count of that

Item, or remove the Item if there are none left. You will also need to update the total cost calculation to become the total value of all your stock.

9. *[Optional]* Read up on how to balance a binary tree and implement a practical solution to improve the balance of your binary tree. Note that you should design and write up the algorithm you intend to implement first before coding it! Be aware that this is a very complex feature to implement so don't consider this feature unless you have implemented all the other optional requirements!

Method

Start your design before the Thursday lab and make use of the assistance available in the lab classes on Thursday, and Friday. You will be building up code that is similar to the Linked List exercises you completed earlier. So, perhaps you can refer back to your own solution to the earlier Linked List Lab(s) or even the sample answers that were provided to you on My Dundee.

In attempting this lab, the following steps could be followed:

1. You will need a class for the **Tree**, and a class for the **TreeNode**. The TreeNode will need to store the Item details and the tree references (left and right). You will also need some kind of **Tester** or **Menu** class as usual too. Design these classes – what variables do you need, and what methods? After you've designed them, implement both of these classes.
2. Develop your code a bit at a time, starting by adding the method that will add a new node into the Tree. Then, use the **Tester** class to add new nodes. Use the debugger to check that the nodes are positioned correctly.
3. Then develop your program to print the nodes "in order" (see the algorithm for this described in the lectures).
4. Expand your program to include a 'search' facility.
5. Expand your program to calculate the total cost by traversing the tree and adding each Item cost to a total cost.
6. Change or replace **Tester** to use a menu – can you re-use one from a previous lab?
7. Add in a main method to your **Tester** class.
8. Produce a test plan for this lab, showing all the situations you need to test for. Run your tests and fill in the results.
9. Design & develop the facility to remove a node from the tree. Develop a test plan – then run your tests & record the results. Hint, there are three scenarios (deleting a leaf, deleting a node with one child and deleting a node with two children). Attempt these scenarios one at a time so that you can at least get one scenario working before you attempting the others. Design these scenarios using pseudocode or a diagram then work out what code to write from your design.
10. Ensure that your Classes and methods all have Javadoc comments and regenerate the Javadoc HTML documentation.
11. Create a JAR file and test your program outside the IDE.

[optional extras]

12. *[Optional]* Add menu options to save the contents of your binary tree to a file and to load the tree from a file.
13. *[Optional]* Implement other forms of traversal for the tree, e.g. preorder and postorder.
14. *[Optional]* Implement full stock counting by adding an Item number field to your TreeNode. Update your *print* method to print out the item count. Update the *add* method to increment the

count if the item exists. Update the total cost calculation. Finally, update the *delete* method to decrement the Item count or delete the Item if the count reaches zero.

15. *[Optional]* Implement a balanced binary tree. NOTE: this will be more challenging! You may wish to search online for examples of how to do this. You may wish to create a separate copy of your project to test with so that your original code doesn't get affected if you don't manage to implement it.

Marking Scheme

Requirements 1-4	Report & Designs (pseudo code and/or flowcharts)	out of 10%
	Test plan and completed test sheets	out of 10%
	Code – menu and tree methods	
	Create a tree & add data to it	out of 20%
	Print a tree and calculate total cost of items	out of 10%
	Find an Item	out of 10%
	Delete an Item	out of 15%
	User interface, including input validation, e.g. tree empty, item already in tree, item not found etc	out of 5%
Requirements 5-7	File reading and writing (7%); other forms of traversal (3%); Item count stock control (10%), a (partially) balanced binary tree (up to 20%)	out of 20%

Please note that well-written code, commenting, a main method and a JAR file have not been specifically allocated marks. This is because it is expected that you will include these in all assignments and that marks will be deducted for any type of bad practice (e.g. poor use of static keyword, lack of commenting or Javadoc comments, badly written code or no runnable JAR file).

Submission

Submit a ZIP file by the deadline stated at the top of this document. Your .zip file will be labelled with your surname, first name and the assignment number, e.g. smith_fred_assignment_2.zip

This file will include:

- A single written report which will include:
  - An introduction stating the problem
  - A summary of the requirements – saying which you were able to tackle and which were successful
  - Your designs using including pseudo code (don't forget your class design) and in particular make sure you explain how you are implementing your tree algorithms.
  - Your test plans & completed test sheets
  - An evaluation of the lab – detailing problems and how you tried to solve them, stating anything you couldn't fix
  - All the JAVA source code files you have used for your assignment (i.e. just include the full eclipse project folder if you are using Eclipse, otherwise ensure that you have included all your java source code files).
  - The Javadoc html, remember to regenerate this after all coding changes have been completed.
  - A runnable JAR file and batch file to run it.

This lab contributes 10% of your coursework marks for AC12001 and 7% for AC22007

Remember to SUBMIT whatever you have done by the deadline, rather than over-running and handing it in late - this should avoid you getting behind and avoid late penalties.