CSCU9A3 Assignment: Real Estate Management System.
Due 4pm, 25th November 2019

You will implement components of a real estate management system, which stores information about properties being marketed by an estate agent (address/name, number of bedrooms, price etc). You will be given partly complete code, and it is your job to complete the code. Comments containing "TODO" mark placeholders in the existing code for you to complete.

Test code is supplied showing you the expected output. Look at the expected output to understand what the code should do. The code for the assignment is available as a zip file on Canvas.

You are required to make the following enhancements to the code, each identified by numbers in square brackets below. The complete assignment is worth 45% of the overall grade for CSCU9A3.

Improving Property Class
The estate agent needs to track interest of potential buyers and tenants so there are not too many viewings in one day for the number of agents available, and the number of years since the last time the property was sold. They also want to add a way of easily controlling whether a property is currently on the market. You are given a basic Property class, but it needs the following functionality added.
[1] Add an integer yearsSinceLastSale attribute to the Property class. Allow the value for yearsSinceLastSale to be read and set via the methods getYearsSinceLastSale() and setYearsSinceLastSale().
[2] Create a new constructor for Property with name, size and yearsSinceLastSale as parameters, and amend the other constructors to set yearsSinceLastSale to -1.
[3] Ensure that addInterested() checks adding new interested buyers/tenants will not exceed the maximum number of daily viewings (maxInterested) before adding them, and returns an appropriate value to confirm whether the interested people could be added.
[4] Add a boolean onMarket attribute to the Property class; initialise it to false in the constructors; and allow it to be read via a method isOnMarket().
[5] Add methods putOnMarket() and takeOffMarket() to set the value of the onMarket attribute

Adding a Farm Class
The manager of the estate agent has decided to expand into managing sales of farms. You need to make an entirely new class which will inherit from the Property class (all farms have the same properties as a Property e.g. a name, size, price), and add some farm-specific functionality.

[6] Using inheritance, add a Farm class that has the protected boolean attribute hasFarmhouse and the protected int attribute numberOfBarns.
[7] Add a constructor that takes parameters name, size, hasFarmhouse and numberOfBarns.
[8] Method toString() to return a string summarising the Farm object. This should be similar to the toString() method in Property and its existing subclasses.

Binary Tree Walk
We need a place to store our collection of Property objects. We will use a Binary Tree that is sorted based on the name of the Property objects it contains. A Node class containing a reference to a Property object has been provided for you together with a partially implemented BinaryTree class consisting of Node objects. The BinaryTree class contains the implementation necessary to add Property objects to the tree and method definitions to walk the tree in an in-order traversal.

RealEstateManager uses this binary tree as an internal attribute called btree and provides a number of public methods to interact with it (addProperty, describePropertyTree, inWalk and find).

[9] Your task is to complete the implementation of the protected inOrder method in BinaryTree.java

You can indirectly make a call to the private traversal methods via the public inWalk method of RealEstateManager mentioned above. Each of these public methods will return a String object containing a comma separated list of strings representing the Properties, by calling getName() on each Property object.

The initial implementation of the protected methods will result in only the root node of the tree being returned. You must add subsequent recursive calls to ensure the tree is visited in the relevant order. As a starting point and to provide some clues as to how you might write your tests, the go method of RealEstateTest.java contains calls to print out the property list, print out the tree (on its side) and calls to each of the traversal method.

Searching a Binary Tree
It is not much use having a more efficient structure for searching for items in a binary tree if we don't actually have a method to find the items. An outline for a find() method has been supplied for you in the class BinaryTree. The go() method in RealEstateTest.java indirectly makes a call to the above private find method and provides code that will determine if the Property with the name 'Barton' or 'Murray' has been located.

[10] Your task is to complete this method such that it finds a Property with the given name or returns null if it couldn't find it.

The RealEstateManager class makes a call to the find() method and provides code that will determine if the Property with the name has been located. You should be able to reuse the logic of your tree walk to recursively implement this method although it will require a little adaptation so that it does not walk the entire tree. Although the solution only involves about 8-10 lines of code, you may want to come back to this task after attempting the next problem. The challenge is handling the recursive search process rather than writing many lines of code (the same is true for the tree walk).

Sorting
To allow the estate agent to properly track its current portfolio, we need to sort the original ArrayList of properties in RealEstateManager by their price and size.

[11] Your task is to implement the InsertionSort algorithm in the RealEstateManager class. The sorting should be based on:

- size or price for each property (*not* using their names via the compareTo() method), depending on the value passed to the "attr" parameter. You can assume this is always a string with value "size" or "price",
- ascending or descending, depending on the value passed to the "asc" parameter.

Outline code to check if the list is sorted correctly, and to print the results of attempting to do this is provided for you in the RealEstateTest class.

Binary Search

One the array is sorted by ascending order of size, it is possible to use a binary search to efficiently search it for a property with a specific size.

[12] Implement the binarySearch method in the RealEstateManager class.

(if you are unable to implement the insertion sort algorithm above, for testing you may use another sorting algorithm to sort the arraylist prior to searching it)

Testing

[13] Write appropriate test methods for the walk and binary search in RealEstateTest.java to check that they work as intended (outline test methods have been provided for you). inOrderTest() should ensure that the walk over the properties in loadProperties() is conducted in the correct order. BinarySearchTest() should ensure that one property is found and one property is not found, assuming the content of the manager is the list of properties in loadProperties().

Summary of Requirements

To summarise, you are required to:

- Modify the Property class (tasks [1]-[5]) (20%)
- Create a Farm class (tasks [6]-[8]) (15%)
- Implement the inOrder and find methods in BinaryTree (tasks [9]-[10]) (15%)
- Implement the insertionSort method in RealEstateManager (task [11]) (20%)
- Implement the binarySearch method in RealEstateManager (task [12]) (10%)
- Write tests for the tree walk and binary search in RealEstateTest (task [13]) (10%)
- Marks will also be allocated for appropriate commenting, sensible variable names, and general quality of code – efficient, concise, readable (10%)

You may also wish to create additional unit tests to the ones required above to ensure that your code is working as intended although these will not be graded.

Submission

Please ensure that you put your student ID (but not your name) at the top of all the .java files and also ensure that all your code compiles. Only a minimal attempt will be made to try to get code to compile if it appears faulty and this will incur a penalty.

Your code will be automatically collected from the directory CSCU9A3\Assignment in your home folder on Monday the 25th of November at 4pm. It is your responsibility to make sure that all the required files are present (with their original names). If you decide to work

away from the University on the assignment, make sure to place your current work back into this folder prior to the collection deadline.

Plagiarism

Work which is submitted for assessment must be your own work. All students should note that the University has a formal policy on plagiarism which can be found at: https://www.stir.ac.uk/about/professional-services/student-academic-and-corporate-services/academic-registry/academic-policy-and-practice/quality-handbook/assessment-and-academic-misconduct/#eight Plagiarism means presenting the work of others as though it were your own. The University takes a very serious view of plagiarism, and the penalties can be severe (ranging from a reduced grade in the assessment, through a fail for the module, to expulsion from the University for more serious or repeated offences). Specific guidance in relation to Computing Science assignments may be found in the Computing Science Student Handbook. We check submissions carefully for evidence of plagiarism, and pursue those cases we find.

Late submission

If you cannot meet the assignment hand-in deadline and have good cause, please see the module coordinator to explain your situation and ask for an extension. Coursework will be accepted up to seven days after the hand-in deadline (or expiry of any agreed extension) but the mark will be lowered by three marks per day or part thereof. After seven days the work will be deemed a non-submission and will receive an X.