Computer Science Fundamentals II Due on Tuesday, December 3, 2019

Assignment 4

CS1027A *University of Western Ontario*

Learning Outcomes

By completing this assignment, you will gain skills in:

- Creating a binary search tree,
- Creating an iterator,
- Using the comparable interface,
- Traversing through a binary search tree, and
- Using a list.

1 Introduction

A new university called University Bloch-Hansen (UBH) is starting up and requires help with setting up their course IT system. The previous IT guy, who was originally going to develop this system, was fired for coming in to work late one too many times! Now UBH is reaching out to **you** to help them create the course structure module in their online infrastructure. In this assignment you will create a Java program that stores course information and allows you to answer queries about that information.

This assignment has two main parts:

- You must create a *binary search tree*. Recall from class that a *binary tree* is a *tree* where each node has at most **2** children. A binary search tree is an *ordered* binary tree; given a node n, all descendants in the left subtree of n have value less than n and all descendants in the right subtree of n have value more than n. You will also need to make a class to represent a binary search tree node and a binary search tree iterator.
- You must create a class to store course information. You must be able to store these course objects in your binary search tree; that is, each node in the tree can hold one course object. Then, you must demonstrate that you can query the binary search tree for specific course information.

You are given an input file specifying the course information and some code that will read from that file and call various methods from your *Course* and *BinarySearchTree* classes. Additionally, you are given some programs that will help you to test and debug your work. The rest of this document will describe in detail the classes that you must create and the code that is provided to you.

2 Classes to Implement

A description of the classes that you need to implement in this assignment is given below. You can implement more classes, if you want. You **cannot** use any static instance variables. You **cannot** use Java's provided *List* class, any of the *Iterator* classes, or any of the other Java classes from the Java library that implements collections.

2.1 BinaryTreeNode.java

This class represents a binary search tree node. The header for this class must be:

- public class BinaryTreeNode<T>

This class must have the following private instance variables:

- *BinaryTreeNode<T> left.* This variable stores a pointer to the left child.
- *BinaryTreeNode<T> right.* This variable stores a pointer to the right child.
- *T element.* This variable stores the data inside the node.

This class needs to provide the following public methods:

- *BinaryTreeNode(T element).* Creates a new node containing the specified element.
- *BinaryTreeNode<T> getLeft().* Returns the left child of this node.
- *void setLeft(BinaryTreeNode<T> left).* Sets the left child of this node.
- *BinaryTreeNode<T> getRight().* Returns the right child of this node.
- *void setRight(BinaryTreeNode<T> right).* Sets the right child of this node.
- *T getElement().* Returns the element contained in this node.
- *void setElement(T element).* Sets the element contained in this node.

You can implement other methods in this class, if you want to, but they must be declared as private.

2.2 BinarySearchTree.java

This class represents a binary search tree. The header for this class must be:

- public class BinarySearchTree<T> implements BinarySearchTreeADT<T>, Iterable<T>

This class must have the following private instance variables:

- *BinaryTreeNode<T> root.* This points to the root of the binary search tree.
- *int size.* This stores the number of nodes in the tree.

This class needs to provide the following public methods:

- *BinarySearchTree().* Creates a new empty binary search tree.
- *BinaryTreeNode<T> find(BinaryTreeNode<T> node, Comparable<T> element) throws NonExistentKeyException.* Returns the node containing *element* or throws an *NonExistentKeyException* if *element* is not in the tree.
- *void add(BinaryTreeNode<T> node, T element) throws DuplicatedKeyException.* Adds *element* in the correct spot in the binary search tree or throws an *DuplicatedKeyException* if *element* is already in the tree.

- *BinaryTreeNode<T> getRoot().* Returns the root node of the binary search tree.
- *boolean isEmpty().* Returns true if the binary search tree is empty, returns false otherwise.
- *int size().* Returns the number of nodes in the binary search tree.
- *boolean contains(T element).* Returns true if *element* is in the binary search tree, returns false otherwise.
- *Iterator<T> iterator().* Returns an iterator over the binary search tree.
- *String toString().* Returns a string representation of the elements in the binary search tree.

You can write more methods in this class, if you want to, but they must be declared as private.

2.3 BinaryTreeIterator.java

This class represents a binary search tree iterator. This iterator **must** return the elements using an *inorder* traversal. The header for this class must be:

- public class BinaryTreeIterator<T> implements Iterator<T>

This class must have the following private instance variables:

- *BinaryTreeNode<T> root.* This points to the root of the binary search tree.
- *T[] items.* This is an array holding the elements of the binary search tree.
- *int current.* This counts how many elements from the binary search tree we have seen.
- *int size.* This stores the number of elements in the binary search tree.

This class needs to provide the following public methods:

- *BinaryTreeIterator(BinaryTreeNode<T> root, int size).* Creates a new binary tree iterator.
- *boolean hasNext().* Returns true if the iterator has another element, returns false otherwise.
- *T next() throws NoSuchElementException.* Returns the next element in the binary search tree. Throws a *NoSuchElementException* if there is no more elements.

You can write more methods in this class, if you want to, but they must be declared as private.

Hint: Make a private helper method that performs an *inorder* traversal that adds nodes to the array *items*. Call this helper method during the constructor. Implement *hasNext()* and *next()* based on the array *items*.

2.4 Course.java

This class represents a course. The header for this class must be:

- public class Course implements Comparable<Course>

This class must have the following private instance variables:

- *String courseCode.* This stores the code of the course.
- *String courseTitle.* This stores the title of the course.
- *int numStudents.* This stores the number of students in the course.
- *String[] professorList.* This stores the professors that teach the course.

This class needs to provide the following public methods:

- *Course(String courseCode, String courseTitle, int numStudents, String[] professorList).* Creates a new course object.
- *String getCourseCode().* Returns the code of the course.
- *String getCourseTitle().* Returns the title of the course.
- *int getNumStudents().* Returns the number of students in the course.
- *String[] getProfessors().* Returns the professors that teach the course.
- *void setCourseCode(String courseCode).* Sets the code of the course.
- *void setCourseTitle(String courseTitle).* Sets the title of the course.
- *void setNumStudents(int numStudents).* Sets the number of students in the course.
- *void setProfessors(String[] professorList).* Sets the professors that teach the course.
- *int compareTo(Course other).* Compare course objects using their course code. This method returns -1 if *this* course comes before the *other* course, returns 1 if *this* course comes after the *other* course, and returns 0 otherwise.
- *String toString().* Returns a string representation of the course.

You can write more methods in this class, if you want to, but they must be declared as private.

You must also create *DuplicatedKeyException.java*, *NonExistentKeyException.java*, and *NoSuchElementException.java*.

3 TreeQuery.java

You must implement the methods in the provided class *TreeQuery.java*. Use the provided method signatures to help you implement the methods. For methods that return variables of type *ListADT*, you may choose what kind of *List* to use. You can use the code from our website's sample code page and from your lab work. Include additional files as necessary to make your choice of *List* compile.

This class must catch any exceptions that are thrown by the methods it calls. For each exception caught, an appropriate message must be printed. The message must explain what caused the exception to be thrown.

4 Command Line Arguments

Registrar.java reads the name of an input file from the command line. You can run the program with the following command:

- *java Registrar nameOfInputFile*

where *nameOfInputFile* is the name of the file containing the course information.

To get Eclipse to supply a command line argument to your program open the "Run -> Run Configurations..." menu item. Make sure that the "Java Application ->Registrar" is the active selection on the left-hand side. Select the "Arguments" tab. Enter the name of the file for the input in the "Program arguments" text box.

5 Classes Provided

You are given *BinarySearchTreeADT.java* to ensure that you implement the correct binary search tree methods. You are given *TestBinarySearchTree.java* to test your binary search tree implementation. You are given *ReadFile.java* to read the course information from a file and create a binary search tree from it. You are given *Registrar.java* to test your query implementations. You are given *ListADT.java*, but you can choose whether to use an *ArrayOrderedList* or an *ArrayUnorderedList* (refer to your labs to see how these work).

6 Sample Input Files Provided

You are given **1** input file that contains course information. For students using Eclipse, put this file in the same folder where the *.classPath* and *.project* files are. If your program can not find the input file, you have not put them in the correct location.

7 Submission

Submit all of your .java files to OWL. **Do not** put the code inline in the textbox. **Do not** submit your *.class* files. If you do this and do not submit your *.java* files your program cannot be marked. **Do not** submit a compressed file with your Java classes (.zip, .rar, .gzip, ...). **Do not** put a "package" command at the top of your Java classes.

8 Non-Functional Specifications

1. **Assignments are to be done individually and must be your own work.** Software will be used to detect cheating.

2. You can assume that the data in the input file is correct. So no error checking of any type is required.

3. Include comments in your code in **javadoc** format. Add javadoc comments at the beginning of your classes indicating who the author of the code is and giving a brief description of the class. Add javadoc comments to the methods and instance variables. Read information about javadoc in the second lab for this course.

4. Include comments in each Java file describing key parts of your program. Comments should be grammatically correct, concise, and easy to understand.

5. Use Java coding conventions and good programming techniques. For example:

   (a) Use Java conventions for naming variables and constants.

   (b) Write readible code: good indentation, appropriate white spaces, etc., are required.

6. Make sure your code runs using Eclipe's Java, even if you do not use Eclipse to write your code.

9 Grading Criteria

- Total Marks: [20]
- Functional Specifications:
  - [1] BinaryTreeNode.java
  - [2] BinarySearchTree.java
  - [1] BinaryTreeIterator.java
  - [1] Course.java
  - [2] TreeQuery.java
  - [5] 20 Binary Search Tree Tests
  - [5] 5 Binary Search Tree Queries
- Non-Functional Specifications:
  - [1] Meaningful variable names, private instance variables
  - [1] Code readability and indentation
  - [1] Code comments

Assignment files (*BinaryTreeNode.java*, *BinarySearchTree.java*, *BinaryTreeIterator.java*, *DuplicatedKeyException.java*, *NonExistentKeyException.java*, *NoSuchElementException.java*, *Course.java*, *TreeQuery.java*, and any files necessary for your choice of *List*) are to be submitted to OWL by 11:55pm on December 3rd (a Tuesday).