CSE2011 - Summer 2014 - Assignment #2

Due Date: 23th of June 2014 at 11:59PM

NOTES

- You may work in groups of up to 3 and make one common submission (if more than one partner submits, it causes us delays and confusion). All partners in the group receive the same mark. All partners should work on all questions; do not simply split the questions into groups, with each partner doing one group of questions. If your partners suddenly decide to quit the course or get ill, it is still your responsibility to submit the assignment on time. Always keep an up-to-date copy of your assignment answers. Assertions that "my partners have the file and I cannot reach them," will not be given consideration. If you are going to work with partners, then the group must submit a file called "partners". The file should contain the cs logins of the partners. Submit this "partners" file with the following command:

  % submit 2011 a2 partners

- Submit the source code for all programs and answers to any questions given in the assignment. The submitted source code is expected to use good coding style and be properly commented. This includes proper javadoc comments (including @param, @returns, etc.). We should be able to produce suitable api's using the javadoc command. Do not place any .java files in a package, i.e. do not use a package statement at the start of your files. Zip all of your .java files and any other files you wish to submit (do not submit any .class files) into a file called a2.zip and submit it using the command

  % submit 2011 a2 a2.zip

QUESTION #1

Say you have an ADT called PQ_Dictionary that has all of the Priority Queue operations listed at the top of page 361 in your text and all of the multiset (bag) operations listed on page 447.

Describe at a high level how you would implement the PQ_Dictionary to achieve worst-case running times of

\Theta(1) for size(), isEmpty(), min()

\Theta(log n) for insert(k, x), removeMin(), find(k), remove(e)

\Theta(n) for findAll(k), entries()

Justify all running times.

QUESTION #2 (simply collect the code from the textbook)

Write all the classes (interface/class/exception) that implement the **NodePositionalList** class with a doubly linked list. NodePositionalList should implement the PositionalList interface on page 275 and should use **DNode** class for double-linking (similar to page 136 but implements the Position interface on page 274):

public class DNode<E> implements Position<E>
public class NodePositionalList<E> implements PositionalList<E>, Iterable<Position<E>>

Please add the following method to check whether the passed position is a valid tree node:

```
protected Position<E> checkPosition(Position<E> v) throws InvalidPositionException
{
if (v == null || !(v instanceof DNode))
   throw new InvalidPositionException("The position is invalid");
return (TreePosition<E>) v;
}
```

NodePositionalList should also implement the Iterable<Position<E>> interface as shown on page 283-287 and should include the methods iterator() and positions():

public Iterator<E> iterator();
public Iterable<Position<E>> positions();

The iterator should return an instance of a class that you need to write:

public class ElementIterator<E> implements Iterator<E>

All your public classes should have a toString() method.
You need to create the needed exception classes, and if you like, to keep things simple, you can make these all runtime exceptions, instead of checked exceptions.
Note that the java.util.Iterator interface has 3 methods hasNext(), next(), and remove() and we don't want to support the remove() method. The way to do this is to have remove() in our implementation just throw an UnsupportedOperationException exception.

Much of the implementation can be found in the text. Feel free to use this or any of the code fragments that come with the text. DO NOT feel free to use any other code that you did not write yourself.

QUESTION #3

Write all the classes (interface/class/exception) that implement the LinkedTree class (as given in the text on page 333 and also explained below) that implements the Tree interface given on page 313. Note that the LinkedTree class uses a NodePositionalList to hold a list of the children of a given node.
Nodes of the tree should be of the TreeNode class (the textbook uses the Node class on page 326 for the binary tree). For general trees we will make a TreePosition interface that extends the Position interface and have the TreeNode class implement that. Here is the TreePosition interface:

```
public interface TreePosition<E> extends Position<E> // inherits element()
{
public void setElement(E o);
public PositionList<Position<E>> getChildren();
public void setChildren(PositionList<Position<E>> c);
public TreePosition<E> getParent();
public void setParent(TreePosition<E> v);
}

public class TreeNode<E> implements TreePosition<E>
{
private E element;  // element stored at this node
private TreePosition<E> parent;  // adjacent node
private PositionList<Position<E>> children;  // children nodes
....
}
```

LinkedTree implements Tree interface and adds some mutator methods to help build the tree, along with a toString method:

```
public class LinkedTree<E> implements Tree<E>
{
protected TreePosition<E> root; // reference to the root
protected int size;        // number of nodes
...
}
```

Please add the following method to check whether the passed position is a valid tree node:

```
protected TreePosition<E> checkPosition(Position<E> v) throws InvalidPositionException
{
if (v == null || !(v instanceof TreePosition))
   throw new InvalidPositionException("The position is invalid");
return (TreePosition<E>) v;
}
```

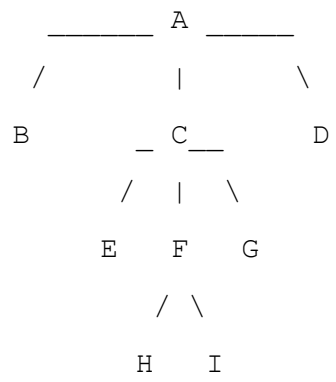
Here are the additional methods not in the Tree interface that you need to implement:

```
/** Adds a root node to an empty tree */
public Position<E> addRoot(E e) throws NonEmptyTreeException { ... }

/** Creates a new tree node in a non-empty tree*/
/** It should put the new Tree Node as the next (i.e. last) child in the children of its parent */
protected TreePosition<E> createNode(E element, TreePosition<E> parent, PositionList<Position<E>>
children) { return new TreeNode<E>(element,parent,children); }

public void swapElements(Position<E> v, Position<E> w) throws InvalidPositionException { ... }
```

If you create a node with no children, that node gets an empty child list, not a null pointer.
You need to create the needed exception classes, and if you like, to keep things simple, you can make these all runtime exceptions, instead of checked exceptions.

Add a toString() method as an instance method. It should return a representation of the tree as given by the parentheticRepresentation method (see the the text). The difference is that the method in the text is static and takes parameters. You may want to use it as a helper method when writing your toString() method.

For example of this method should return A (B, C (E, F (H, I), G), D) for the tree:



Here is a simple tester program:

```
public class Tester
{
   public static void main(String[] args)
   {
      // make empty tree
      LinkedTree<Character> T = new LinkedTree();

      // add root
      T.addRoot('A');
```

```
        // add children of root
        T.createNode('B', (TreeNode)(T.root()), new NodePositionList());
        TreePosition  C = T.createNode('C', (TreeNode)(T.root()), new NodePositionList());
        T.createNode('D', (TreeNode)(T.root()), new NodePositionList());

        // add children of node C
        T.createNode('E', C, new NodePositionList());
        TreePosition  F = T.createNode('F', C, new NodePositionList());
        T.createNode('G', C, new NodePositionList());

        // add children of node F
        T.createNode('H', F, new NodePositionList());
        T.createNode('I', F, new NodePositionList());

        // print out info about the tree
        System.out.println("Size = " + T.size());
        System.out.println("Here is the tree:");
        System.out.println(T);
    }
}
```

This should compile and the tree it prints out should look like:

A (B, C (E, F (H, I), G), D)

A lot of code can be found in the text. Feel free to use this or any of the code fragments that come with the text. DO NOT feel free to use any other code that you did not write yourself.