Tree Implementation with Linked Nodes

Reading next LDC Ch 17.1, 17.2
Tree Implementation with Linked Nodes

- Nodes contain a reference to the data stored in the node, and references for each of the possible children of the node
  - **binary** tree: 2 references required – left and right children
  - **n-ary** tree: n references required – one for each possible child

- Each **binary tree** node is defined using a class **BTNode** – (similar to **LinearNode** used in linked lists)

- Trees lend themselves to recursive processing for many operations
package javafoundations;

import java.util.Iterator;

public interface BinaryTree<T> extends Iterable<T> {
    // Returns the element stored in the root of the tree.
    public T getRootElement();

    // Returns the left subtree of the root.
    public BinaryTree<T> getLeft();

    // Returns the right subtree of the root.
    public BinaryTree<T> getRight();

    // Returns true if the binary tree contains an element that matches the specified element and false otherwise.
    public boolean contains(T target);
} (more...)
// Returns reference to the element in tree matching target.  
public T find (T target);

// Returns true if the binary tree contains no elements, false o/w  
public boolean isEmpty();

// Returns the number of elements in this binary tree.  
public int size();

// Returns the string representation of the binary tree.  
public String toString();

// Returns a preorder traversal on the binary tree.  
public Iterator<T> preorder();

// Returns an inorder traversal on the binary tree.  
public Iterator<T> inorder();

// Returns a postorder traversal on the binary tree.  
public Iterator<T> postorder();

// Performs a level-order traversal on the binary tree.  
public Iterator<T> levelorder();
}
A possible set of operations for a binary tree is shown in the `BinaryTree` interface implemented by `LinkedBinaryTree`.

- `BinaryTree` has no methods to add a particular element, or to remove a particular element from the tree (you build it bottom-up).

- Before we show class `BTNode`, and class `LinkedBinaryTree` let’s show an application: **Decision Tree** of an expert system (AI).
16.5 – Decision Trees

- A **decision tree** is a tree whose nodes represent decision points, and whose children represent the options available.

- The **leaves** of a decision tree represent the possible **conclusions** that might be drawn based on the answers.

- Decision trees are used in **expert systems** – software that attempts to represent the knowledge of an expert in a particular field.

- Decision tree with yes/no binary tree.

- Expertise **examples**
  - a doctor
  - a car mechanic
  - accountant
  - PC help desk!?!?

> run BackPainAnalyzer
So, you're having back pain.
Did the pain occur after a blow or jolt?
  n

Do you have a fever?
  n

Do you have persistent morning stiffness?
  y

You may have an inflammation of the joints.
16.5 – A Decision Tree for Diagnosing Back Pain

Did the pain occur after a blow or jolt?

- **N**
  - Do you have a fever?
    - **N**
      - Do you have persistent morning stiffness?
        - **N**
          - See doctor if pain persists.
        - **Y**
          - You may have an inflammation of the joints.
    - **Y**
      - Do you have a sore throat or runny nose?
        - **N**
          - See doctor to address symptoms.
        - **Y**
          - You may have a respiratory infection.
  - **Y**
    - Do you have difficulty controlling your arms or legs?
      - **N**
        - Do you have pain or numbness in one arm or leg?
          - **N**
            - You may have a sprain or strain.
          - **Y**
            - You may have a muscle or nerve injury.
      - **Y**
        - Emergency! You may have damaged your spinal cord.

- **Y**
  - Do you have a fever?
    - **N**
      - Do you have persistent morning stiffness?
        - **N**
          - See doctor if pain persists.
        - **Y**
          - You may have an inflammation of the joints.
    - **Y**
      - Do you have a sore throat or runny nose?
        - **N**
          - See doctor to address symptoms.
        - **Y**
          - You may have a respiratory infection.
  - **Y**
    - Do you have difficulty controlling your arms or legs?
      - **N**
        - Do you have pain or numbness in one arm or leg?
          - **N**
            - You may have a sprain or strain.
          - **Y**
            - You may have a muscle or nerve injury.
      - **Y**
        - Emergency! You may have damaged your spinal cord.

The left child represents the answer “No”
The right child represents the answer “Yes”
```java
public class BackPainAnalyzer
{
    public static void main (String[] args)
    {
        BackPainExpert expert = new BackPainExpert();
        expert.diagnose();
    }
}
```
import javafoundations.*;

public class BackPainExpert
{
    private LinkedBinaryTree<String> tree;

    public BackPainExpert()
    {
        String e1 = "Did the pain occur after a blow or jolt?";
        String e2 = "Do you have a fever?";
        String e3 = "Do you have difficulty controlling your arms or legs?";
        String e4 = "Do you have persistent morning stiffness?";
        // (etc)
        n7 = new LinkedBinaryTree<String>(e7);
        n2 = new LinkedBinaryTree<String>(e2, n4, n5);
        n3 = new LinkedBinaryTree<String>(e3, n6, n7);
        tree = new LinkedBinaryTree<String>(e1, n2, n3);
    }
}
// Follows the diagnosis tree based on user responses.

public void diagnose()
{
    Scanner scan = new Scanner(System.in);
    LinkedBinaryTree<String> current = tree;

    System.out.println("So, you're having back pain.");
    while (current.size() > 1)
    {
        System.out.println(current.getRootElement());
        if (scan.nextLine().equalsIgnoreCase("N"))
            current = current.getLeft();
        else
            current = current.getRight();
    }

    System.out.println(current.getRootElement());
}
A possible set of operations for a binary tree is shown in the `BinaryTree` interface implemented by `LinkedBinaryTree`.

`BinaryTree` has no methods to add a particular element, or to remove a particular element from the tree.

First, we show class `BTNode`, then class `LinkedBinaryTree`.
package javafoundations;

cpyublic class BTNode<T> {
    protected T element;
    protected BTNode<T> left, right;

    public BTNode (T elmt) {
        element = elmt;
        left = right = null;
    }
}
16.4 - java foundations

BTNode

// Returns the element stored in this node.
public T getElement()
{ return element; }

// Sets the element stored in this node.
public void setElement (T element)
{ this.element = element; }

// Returns the left subtree of this node.
public BTNode<T> getLeft()
{ return left; }

// Sets the left child of this node.
public void setLeft (BTNode<T> left)
{ this.left = left; }

// Returns the right subtree of this node.
public BTNode<T> getRight()
{ return right; }

// Sets the right child of this node.
public void setRight (BTNode<T> right)
{ this.right = right; }

(more...)
Contains diagram with nodes

```java
public BTNode<T> find (T target) {
    BTNode<T> result = null;
    if (element.equals(target))
        result = this;
    else {
        if (left != null)
            result = left.find(target);
        if (result == null && right != null)
            result = right.find(target);
    }
    return result;
}
```

(more...)
public int count() {
    int result = 1;
    if (left != null)
        result += left.count();
    if (right != null)
        result += right.count();
    return result;
}

int total = cs.count()
```java
// Performs an inorder traversal on this subtree, updating the specified iterator.
public void inorder (ArrayIterator<T> iter)
{
    if (left != null)
        left.inorder (iter);

    iter.add (element);

    if (right != null)
        right.inorder (iter);
}

// The following methods are left as programming projects during sleep
public void preorder (ArrayIterator<T> iter) { }
public void postorder (ArrayIterator<T> iter) { }
```
An iterator is an object that provides a means of processing a collection of objects, one at a time.

**Interface Iterator**

<table>
<thead>
<tr>
<th>Method Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>boolean hasNext()</td>
</tr>
<tr>
<td>Returns true if the iteration has more elements.</td>
</tr>
<tr>
<td>E next()</td>
</tr>
<tr>
<td>Returns the next element in the iteration.</td>
</tr>
<tr>
<td>void remove()</td>
</tr>
<tr>
<td>Removes from the underlying collection the last element returned by the iterator (optional operation).</td>
</tr>
</tbody>
</table>

By implementing the Iterator interface, a class formally establishes that: objects of that type are iterators.

Now, the **for-each** version of the **for** loop can be used to process the items in the iterator.
implementing an iterator using array

// ArrayIterator.java  Java Foundations
// Represents an iterator over the elements of a collection.
// ******************************************************************************
import java.util.*;

public class ArrayIterator<T> implements Iterator<T>
{
    private int DEFAULT_CAPACITY = 10;
    private int count;  // the number of elements in the iterator
    private int current;  // the current position in the iteration
    private T[] items;  // the iterator's storage for elements

    //---------------------------------------------------------------------------
    //  Sets up this iterator.
    //---------------------------------------------------------------------------
    public ArrayIterator()
    {
        items = (T[]) (new Object[DEFAULT_CAPACITY]);
        count = 0;
        current = 0;
    }
}(more...)
public void add (T item) {
  if (count == items.length)
    expandCapacity();
  items[count] = item;
  count++;
}

private void expandCapacity() {
  T[] larger = (T []) (new Object[items.length*2]);

  int location = 0;
  for (T element : items)
    larger[location++] = element;

  items = larger;
}
public boolean hasNext()
{
    return (current < count);
}

public T next()
{
    if (! hasNext())
        throw new NoSuchElementException();
    current++;
    return items[current - 1];
}

public void remove() throws UnsupportedOperationException
{
    throw new UnsupportedOperationException();
}
16.4 – A Binary Tree Implementation

- A possible set of operations for a binary tree is shown in the BinaryTree interface.
- BinaryTree has no methods to add a particular element, or to remove a particular element from the tree.
- Refined versions of binary tree (such as binary search trees) will define those methods based on specific characteristics.
- BinaryTree is still useful in certain situations.
- First we show class BTNode, then class LinkedBinaryTree.
package javafoundations;

import java.util.Iterator;
import javafoundations.*;
import javafoundations.exceptions.*;

public class LinkedBinaryTree<T> implements BinaryTree<T> {
    protected BTNode<T> root;

    // Creates an empty binary tree.
    public LinkedBinaryTree() {
        root = null;
    }

    // Creates a binary tree with the specified element as its root.
    public LinkedBinaryTree(T element) {
        root = new BTNode<T>(element);
    }

    // (more...)
}
public LinkedBinaryTree (T element, LinkedBinaryTree<T> left, LinkedBinaryTree<T> right) {
    this.root = new BTNode<T>(element);
    this.root.setLeft(left.root);
    this.root.setRight(right.root);
}

(more...)
public LinkedBinaryTree<T> getLeft() {
    if (root == null)
        throw new EmptyCollectionException("Get left failed. Tree is empty.");
    LinkedBinaryTree<T> result = new LinkedBinaryTree<T>();
    result.root = this.root.getLeft();
    return result;
}

(more...)
16.4 – javafoundations.LinkedBinaryTree

```java
// Returns the element stored in the root of the tree. Throws an
// EmptyCollectionException if the tree is empty.

public T getRootElement()
{
    if (root == null)
        throw new EmptyCollectionException("Get root failed. Tree is empty");
    return root.getElement();
}

// Returns the element in this binary tree that matches the
// specified target. Throws a ElementNotFoundException if the
// target is not found.

public T find (T target)
{
    BTNode<T> node = null;
    if (root != null)
        node = root.find(target);
    if (node == null)
        throw new ElementNotFoundException("Find operation failed. " + "No such element in tree.");
    return node.getElement();
}
```

(more...)
public int size()
{
    int result = 0;
    if (root != null)
        result = root.count();
    return result;
}

public Iterator<T> inorder()
{
    ArrayIterator<T> iter = new ArrayIterator<T>();
    if (root != null)
        root.inorder (iter);
    return iter;
}
// Populates and returns an iterator containing the elements in
// this binary tree using a levelorder traversal.

public Iterator<T> levelorder()
{
    LinkedQueue<BTNode<T>> queue = new LinkedQueue<BTNode<T>>();
    ArrayIterator<T> iter = new ArrayIterator<T>();

    if (root != null)
    {
        queue.enqueue(root);
        while (!queue.isEmpty())
        {
            BTNode<T> current = queue.dequeue();

            iter.add (current.getElement());

            if (current.getLeft() != null)
                queue.enqueue(current.getLeft());
            if (current.getRight() != null)
                queue.enqueue(current.getRight());
        }
    }

    return iter;
}

(more...)
You also need the iterator() method!

```java
public Iterator<T> iterator()
{
    return inorder();
}
```

The following methods are left as programming projects.

```java
// public LinkedBinaryTree<T> getRight() { }
// public boolean contains (T target) { }
// public boolean isEmpty() { }
// public String toString() { }
// public Iterator<T> preorder() { }
// public Iterator<T> postorder() { }
```

**IMPORTANT:**
To test this file you need to create a driver OUTSIDE javafoundations. You cannot do it by adding a main() method in LinkedBinaryTree.java
Is this clear?

I’ll pause for a moment so you can let this information sink in.