```
Script started on 2021-04-11 13:17:13-0500
l_ladios@ares:~$ cat BinaryTree.info
 /**************************************************************************
     *                                                                        *
     *  NAME:  Leia Ladios                        CLASS:  CSC121-W01          *
     *                                                                        *
     *  Assignment: Lab P-8.61                    Level:   2                  *
     *                                                                        *
     *  Description:                                                          *
     *                                                                        *
        P-8.61: Implement the binary tree ADT using the array-based
        representation described in Section 8.3.2
     *                                                                        *
    **************************************************************************/


l_ladios@ares:~$ cat Iterable.java

public interface Iterable<T> {

        Iterator<T> iterator();
}

l_ladios@ares:~$ javac Iterable.java
l_ladios@ares:~$ cat Iterator.java


import java.util.NoSuchElementException;

public interface Iterator<T> {

        boolean hasNext();

        T next() throws NoSuchElementException;

        void remove(T t) throws IllegalStateException;
}

l_ladios@ares:~$ javac Iterator.java
l_ladios@ares:~$ cat Tree.java

import java.util.NoSuchElementException;

public interface Tree<T> extends Iterable<T>{


        T root();

        int parent(int index) throws IllegalArgumentException;

        Iterable<T> children(int index) throws IllegalArgumentException;

        int numChildren(int index) throws IllegalArgumentException;

        boolean isInternal(int index) throws IllegalArgumentException;

    boolean isExternal(int index) throws IllegalArgumentException;

    boolean isRoot(int index) throws IllegalArgumentException;

    int size();

    boolean isEmpty();

    Iterator<T> iterator();

    Iterable<T> positions();

}

l_ladios@ares:~$ javac Tree.java
l_ladios@ares:~$ cat AbstractTree.java

public abstract class AbstractTree<T> implements Tree<T> {

        public boolean isInternal(int index) {
                return numChildren(index) > 0;
        }

        public boolean isExternal(int index) {
                return numChildren(index) == 0;
        }

        public boolean isRoot(int index) {
                return index == 0;
        }

        public boolean isEmpty() {
                return size() == 0;
        }
}

l_ladios@ares:~$ javac AbstractTree.java
l_ladios@ares:~$ cat BinaryTree.java


import java.util.NoSuchElementException;

public interface BinaryTree<T> extends Tree<T>{

        T left(int index) throws IllegalArgumentException;

        T right(int index) throws IllegalArgumentException;

        int sibling(int index) throws IllegalArgumentException;

}
```

```
l_ladios@ares:~$ javac AbstractBinaryTree.java
Note: AbstractBinaryTree.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.
l_ladios@ares:~$ cat ArrayBinaryTree.java


import java.util.ArrayList;
import java.util.NoSuchElementException;

public class ArrayBinaryTree<T> extends AbstractBinaryTree{

        private T root = null;
        private int size = 0;
        private ArrayList<T> treeArr = (ArrayList<T>) new ArrayList<String>();

        public class ArrayTreeIterator implements Iterator<T> {

                private int i = 0;
                private boolean removable = false;
                private T current;


                @Override
                public boolean hasNext() {
                        return i < size;
                }

                @Override
                public T next() throws NoSuchElementException {
                        if(i == size)
                                throw new NoSuchElementException();
                        removable = true;
                        return treeArr.get(i++);
                }

                @Override
                public void remove(T t) throws IllegalStateException {
                        if(!removable)
                                throw new IllegalStateException("There is nothing t
                        treeArr.remove(i - 1);
                        i--;
                        removable = false;

                }

        }

        /**
         * constructs empty binary tree
         */
        public ArrayBinaryTree() {

        }

        public void addRoot(T element) {
                root = element;
                treeArr.add(0, root);
                size ++;
        }

        public void addLeft(int index, T element) throws IllegalArgumentException {
                try {

                        if(left(index) != null) {
                                System.out.println("This index already has a left c
                        }
                }
                catch(IllegalArgumentException e) {
                        treeArr.add(2 * (index) + 1, element);
                        size++;

                }
        }

        public void addRight(int index, T element) throws IllegalArgumentException{
                try {
                        if(right(index) != null)
                                System.out.println("This index already has a right
                }
                catch(IllegalArgumentException e) {
                        treeArr.add(2 * (index) + 2, element);
                        size++;
                }
        }

        @Override
        public T left(int index) throws IllegalArgumentException {
                if(this.size() <= 2 *(index) + 1) { // no left children
                        throw new IllegalArgumentException("This index does not hav
                }

                return treeArr.get(2 * (index) + 1); //return left child if it exis
        }

        @Override
        public T right(int index) throws IllegalArgumentException {
                if(this.size() <= 2 * (index) + 2) { // no left children
                        throw new IllegalArgumentException("This index does not hav
                }
                return treeArr.get(2 * (index) + 2); //return right child if it ex


        }

        @Override
        public Object root() {

                if(this.isEmpty())
```

```
                return null;
        else
                root = treeArr.get(0);
        return root;
}

@Override
public int parent(int index) throws IllegalArgumentException {

        if(index == 0) {
                throw new IllegalArgumentException();
        }

        return (index -1) / 2;


}

@Override
public Iterable<T> children(int index) throws IllegalArgumentException {

        if(this.isExternal(index)) {
                throw new IllegalArgumentException();
        }
        else {
            return this.children(index);
        }
}

@Override
public int numChildren(int index) throws IllegalArgumentException {

        int count = 0;
        if(isEmpty()) {
                throw new IllegalArgumentException();
        }
        else {
                int parentIndex = parent(index);
                if(2 * (index) + 1 > parentIndex){
                        count++;
                }
                if(2 * (index) + 2 > parentIndex) {
                        count++;
                }
                return count;
        }


}

@Override
public int size() {

        return size;
        }

        @Override
        public Iterator<T> iterator() {

                return new ArrayTreeIterator();
        }

        @Override
        public Iterable positions() {
                // TODO Auto-generated method stub
                return null;
        }

//      @Override
//      public Iterable<T> positions() {
//              //default iteration method
//              return preOrder();
//      }
//
//      private Iterable<T> preOrder() {
//              ArrayList<T> iterateMe = new ArrayList<T>();
//
//              iterateMe.add(0, (T) new String("Hello"));
//              return (Iterable<T>) iterateMe;
//      }
//
//      public void preOrderSubtree(int index, ArrayList<T> snap) {
//              snap.add(treeArr.get(index));
//
//              this.children(index);
//      }



}

l_ladios@ares:~$ javac ArrayBinaryTree.java
Note: ArrayBinaryTree.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.
l_ladios@ares:~$ cat BinaryTreeMain.java

import java.util.ArrayList;

public class BinaryTreeMain {

        public static void main(String[] args) {
                ArrayBinaryTree biTree = new ArrayBinaryTree();

                biTree.addRoot(new String ("4"));

                biTree.addLeft(0, new String("5"));

                biTree.addRight(0, new String("6"));
```

```
                biTree.addLeft(1, new String("7"));


                Iterator it = biTree.iterator();

                while(it.hasNext()) {
                        System.out.println(it.next());
                }

                for(String str: biTree) {
                        System.out.println(str);
                }

        }
}


l_ladios@ares:~$ javac BinaryTreeMain.java
BinaryTreeMain.java:24: error: for-each not applicable to expression type
                for(String str: biTree) {
                                ^
  required: array or java.lang.Iterable
  found:    ArrayBinaryTree
Note: BinaryTreeMain.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.
1 error
l_ladios@ares:~$ java BinaryTreeMain
Error: Could not find or load main class BinaryTreeMain
l_ladios@ares:~$ exit
exit

Script done on 2021-04-11 13:20:30-0500
```