package edu.duke;

import java.util.ArrayList;

import java.util.Arrays;

import java.util.List;

/\*\*

\* The <code>StorageResource</code> class stores any number of <code>String</code> objects and

\* allows access to these stored values one at a time, using the method <code>data</code>. These

\* strings can then be iterated over in the order they were added using a <code>for</code> loop.

\*

\* <P>

\* This class mirrors an <code>ArrayList&lt;String&gt;</code> in some functionality, but is simpler

\* to use and fits the Duke/Coursersa model of creating and using iterables.

\*

\* <P>

\* Example usage:

\*

\* <PRE>

\* FileResource fr = new FileResource();

\* StorageResource store = new StorageResource();

\* for (String s : fr.words()) {

\* store.add(s);

\* }

\* // can process store here, e.g.,

\* // get number of strings stored

\* int x = store.size();

\* for (String s : store.data()) {

\* // print or process s

\* }

\* </PRE>

\*

\* <P>

\* This software is licensed with an Apache 2 license, see

\* http://www.apache.org/licenses/LICENSE-2.0 for details.

\*

\* @author Duke Software Team

\*/

public class StorageResource {

private List<String> myStrings;

/\*\*

\* Create an empty <code>StorageResource</code> object

\*/

public StorageResource () {

myStrings = new ArrayList<String>();

}

/\*\*

\* Create a <code>StorageResource</code> object, loaded with the Strings passed as parameters.

\*/

StorageResource (String... data) {

myStrings = new ArrayList<String>(Arrays.asList(data));

}

/\*\*

\* Create an <code>StorageResource</code> object that is a copy of another list.

\*

\* @param other the original list being copied

\*/

public StorageResource (StorageResource other) {

myStrings = new ArrayList<String>(other.myStrings);

}

/\*\*

\* Remove all strings from this object so that <code>.size() == 0</code>.

\*/

public void clear () {

myStrings.clear();

}

/\*\*

\* Adds a string to this storage object.

\*

\* @param s the value added

\*/

public void add (String s) {

myStrings.add(s);

}

/\*\*

\* Returns the number of strings added/stored in this object.

\*

\* @return the number of strings stored in the object

\*/

public int size () {

return myStrings.size();

}

/\*\*

\* Determines if a string is stored in this object.

\*

\* @param s string searched for

\* @return true if and only if s is stored in this object

\*/

public boolean contains (String s) {

return myStrings.contains(s);

}

/\*\*

\* Create and return an iterable for all strings in this object.

\*

\* @return an <code>Iterable</code> that allows access to each string in the order stored

\*/

public Iterable<String> data () {

return myStrings;

}

}

package edu.duke;

import java.util.Iterator;

/\*\*

\* This utility class allows multiple classes to iterate over a text source in

\* multiple ways.

\*

\* It also serves to show how to implement an iterator.

\*

\* @author Duke Software Team

\*

\* This software is licensed with an Apache 2 license, see

\* http://www.apache.org/licenses/LICENSE-2.0 for details.

\*/

class TextIterable implements Iterable<String> {

private String[] myStrings;

/\*\*

\* Create from a given string.

\*/

public TextIterable (String source, String regexp) {

myStrings = source.split(regexp);

}

/\*\*

\* @see java.lang.Iterator

\*/

@Override

public Iterator<String> iterator () {

return new Iterator<String>() {

private int myCount = 0;

@Override

public boolean hasNext () {

return myCount < myStrings.length;

}

@Override

public String next () {

String s = myStrings[myCount];

myCount++;

return s;

}

};

}

}