COMP1161 Lab 4 – Week of February 11

Objectives:

This lab will give you an introduction to :

- Objects that can be used to store collections of objects (a list).
- Objects that can provide methods to iterate through a collection of objects (Iterators)

Topics to be covered:

- The `ArrayList` class as an example of an object that can store collections of things (objects).
- Using `the Scanner` class to read data from a text file.

At the end of this lab you should be able to:

1. Declare an ArrayList object;
2. Instantiate an ArrayList object;
3. Explain the use of generics in the declaration and instantiation of an ArrayList object;
4. Add an item to an ArrayList object.
5. Loop through the items in an ArrayList object and process each item in the list.
6. Process data froma text file using the `File` and `Scanner` classes.

Preamble

Many of the programs that you write will need to manage collections of objects. For example, if you are writing an application to track the weather you will very likely need to keep a collection of readings (one for each day). The same will be true for a library system, or a student registration system. Java has several classes that are designed for managing collections. Of these the `ArrayList` class is perhaps the easiest one to use. An `ArrayList` is simply a structure that can keep a list of items and provides methods that you can use to manipulate the items.

We will write an application that manages a collection of words. The application will use a class named `Glossary` whose constructor reads words from a text file and stores the words in an `ArrayList` object. The `Glossary` class will also provide methods to count the number of times a given word appears in the list, and a `toString` method that formats the list of words in a pleasing manner.

Lab Activities

- Create the class `Glossary` in the package lab4. As usual, you are to remove the code that BlueJ has generated leaving just the empty class body.

The `ArrayList` class is part of the `java.util` package of the standard Java class library.

- Add the necessary code to your class to import the required package.

A glossary stores a list of words. We will write the `Glossary` class to do this. The class will manage the list of words, so we will create a variable for the list. Let us call it `words`

The variable `words` is a reference to an object of type `ArrayList` which will be used to store the collection of words in the glossary. This variable is an instance variable As such its declaration must be made in the class body but not within a method body. Because an `ArrayList` object will itself contain other objects, the declaration needs to state what type of objects these will be. In this case, the list will store strings. The declaration is as follows:

```
private ArrayList<String> words;
```

- Declare the variable `words` as shown. Compile your class when you have done just to make sure all syntax is correct.

The construct `<String>` (referred to as a generic) is used to declare what type of object the list will store. If there was a class called `Name` and the list was intended to store objects of that type then the declaration would have been:

```
private ArrayList<Name> …..;
```

Ultimately the `Glossary` class will load its list of words from a file. However, let us first create a simple constructor that does some basic operations on the list of words. This will be the default constructor. We will use it to introduce the use of the `ArrayList` class.

- Declare a constructor, one that does not accept an argument. Your code should look something like the following:

```
import java.util.ArrayList;

public class Glossary {

      private ArrayList<String> words;   //The collection

      public Glossary(){ //Default Constructor

      } //end of default constructor;

} //end of class body
```

Now let us work on the constructor. The constructor instantiates the object `words` uing the `new` keyword just as for any other class. However, for an `ArrayList` object, the type of object that the list will store must be stated in the instantiation as follows:

```
words = new ArrayList<String>(); //Create the empty list
```

- Go ahead and include this statement in your constructor. Again, you should compile the class just to make sure that all syntax is correct

The empty parentheses at the end of the statement (very strange looking but absolutely necessary) indicates that we are using the default constructor of the `ArrayList` class. Actually there are quite a few other constructors for the `ArrayList` class but this default will do for now.

Adding Items at end of the list

Items are added to the end of the list by using the `add` method. This method is specified as follows.

`public boolean add (E obj);`
- Adds an item to the end of the list

The parameter `obj` is an object of the same type (E) as the type that the list stores. An example is given below:

```
words.add("zebra");
```

This statement adds the word "zebra" to the end of the list `words`.

- Write code to add up to five words of your choice to the list of words.

Inserting Items into the List

Each item in a list has a position (an index) with the first item at index 0, the second at index 1, and so on. There is a second version of the `add` method which allows you to specify where in the list an item is to be added. The method is specified as follows:

`public void add (int index, E obj)`
- Inserts an item into the list at the position given by the argument index. The first position is 0.

Example: `words.add(3, "Third");`

This example inserts the word "Third" at index position 3 shifting the rest of the list to the right.

- Add a statement to insert the word “alpha at the front of the list.

Inspecting an Object with BlueJ

BlueJ provides a feature that allows you to inspect an obect. Here is an example of how it is used.

- Close the editor and go back to the window in which the classes in your package are displayed.
- Right click on the `Glossary` class and select the constructor method.
- Run the constructor method. You will see a `Glossary` object displayed at the bottom of the window.
- Double click on the object to dislay a window that shows the internals of the object including the variable `words`. The variable is a pointer (as shown).
- Double click on the pointer to see the list of strings that the variable `words` references. You should see the five words that you added to the list.

You can use this feature to observe the state of an object (its values) as you invoke its metohds. Try the following:

- Change the code in the constructor to vary the number of words added to the list. Also, try to add new words at various positions. Use the BlueJ inspector to ensure that the list is updated correctly each time.

A `toString()` method

Now let us write a `toString` method for the `Glossary` class so that whenever a `Glossary` object is printed the list of words it contains will be displayed. Remember, an object’s `toString` method is automatically invoked whenever we use the name of the object in a print, or println statement, or in any situation where a string is required. Recall that the `toString` method is declared as follows:

```
public String toString() {
//code goes here. Must have a return statement.
}
```

Let us write a simple `toString` method for now by using the the `toString` method of the `ArrayList` class. The statement to do this is as follows

```
return words.toString();
```

- Write the `toString` method for the `Glossary` class using the statement given above.

A Test Driver

Create a new class (call it `TestGlossary`) that contains code to test the `Glossary` class. This class will contain a `main` method with the following statements.

```
Glossary myGlossary = new Glossary(); //create a Glossary object
System.out.println(myGlossary); //Display the object
```

This is an example of a driver class. It exists only to provide a main method which you can use to test the operations of one, or more classes. In a way it is an application. After all, it has a `main` method.

- Run the driver class and observe the results.

Other Methods of the ArrayList Class

There are several other methods that are defined for the ArrayList class. Here are a few:

`public boolean remove(int index)`
- removes the item at the position given by index.

`public boolean isEmpty()`
- returns the value **true** if the list is empty.

`public int size()`
- returns the count of the number of items in the list.

`public Object get (int index)`
- Fetch and return the item at position given by the argument index. The item is returned as an object.

Fetching Items from an ArrayList

The items in a list can be retrieved in several ways. The most straightforward approach is to use the `get` method. Her is an example of the use of this method:

Example: `String s1 = words.get(2);`

The example will fetch the item at index position 2 and assign the variable `s1` to reference this object.

The `get` method can be used with a simple `while` loop to start at the beginning of the list and to "step through" the list processing each item. For example, you might print all of the words in the glossary with the following algorithm:

```
set i = 0;
while i  < length of the list
        get item at position i
        print item
```

- Modify the `toString` method so that the words in the glossary are printed one per line when the `Glossary` object is printed by the driver class. [*Hint: You can do this by adding a newline character ("`\n`") between the words in the string that the `toString` method returns.*]

Building a glossary from a text file

We will now improve the `Glossary` class so that it can build the list of words using data that is kept in a text file. First, however, we must understand how a Java program reads data from a text file.

The class `File` is used in a Java program to open a file. This class is in the package `java.io`.

To open a file, you declare a variable of type `File` and instantiate it, passing the name of the file as an argument. For example,

```
File myFile = new File(name-of-file);
```

A `Scanner` object can be used to read from a file just the same as it is used to read from the keyboard. This is done as follows:

```
Scanner scanner-object = new Scanner(myFile);
```

or simply:

```
Scanner scanner-object = new Scanner(new File(name-of-file));
```

For example, to open a file named "glossary.txt" you would use the statement:

```
Scanner scan = new Scanner(new File("glossary.txt"));
```

The `File` class is a part of the `java.io package` so you must import this package.

- Declare the second constructor and write code to open a file. The constructor is to accept the name of the file as its single parameter. Compile the class to ensure that you have followed all syntax rules but do not attempt to run the code yet.

Java has a special type, the `Iterator` class, which provides methods that can be used to move through a collection of items. Once an `Iterator` has been associated with a collection it (the `Iterator`

object) can be used to travel through a list (or other collection) starting at the beginning and going on to the end.

An `Iterator` object always provides at least two methods. These are:

`hasNext()`
- returns the value **true** if there are more items in the collection to be visitied;

`next()`
- returns the next item in the list.

A scanner object is an `Iterator` object. It provides the methods `hasNext()` and `next()`, in addition to the methods you already know (e.g. `nextInt()`, `nextDouble()`, etc.).

The `hasNext()` method returns the value **true** as long as there is more data to be read from the file. This method can be used in a `while` loop to read data from a file as follows:

```
String s1;
while scan.hasNext()
{
     s1 = scan.nextLine();
     // do stuff with s1
}
```

Now let us complete the task of writing a constructor that reads, from a file, the words that will be stored in the glossary.

- Download the file **lab4.zip** from the course web site and extract the file **glossary.txt.**. Ensure that the text file is placed in the package folder.

- Use a text editor such as Wordpad, or Notepad to examine the file. The file contains one word per line. These words are to be loaded by the `Glossary` object.

- Complete the constructor by adding code to read each line of data (a word) from the file and add each word to the collectionof words.

- Use the driver class to test the new constructor.

If all has gone well you now have a class that can be used to create a list of words by reading the words from a file. We can create some interesting methods to perform operations on the list of words.

- Write a method named `occurrences` that accepts a string as its parameter. The parameter is a word. The method should return an integer being the number of occurrences of the word in the glossary.

- Write a method named `prune` that accepts a string as its parameter. The parameter is a word. The method should remove all occurrences of the word from the glossary and

should return an integer value to indicate how many occurrences were found and removed..

- Modify the driver class to use the new methods.
- Demonstrate your program to a lab tech.
- Submit a **jar** file with your program. Remember to include the source files.

Discussion

Suppose the text file had several words in each line. Would your class work properly? Please say why, or why not and (if not) describe briefly how the problem could be addressed.

**** End of Lab ****