Laboratory 5 – Week 5

## Data Structures

## 5.1 Introduction

**Firstly, the content of this worksheet is (*may be*) part of the Task #1a, Task #1b and Task #2 assessments.**

This laboratory involves the creation of a number of Java programs that perform simple tasks that manipulate data structures. Make sure that you save any code you write. Also make sure you save any results or notes that you observe about your work.

This laboratory looks at four of the data structures that we covered in the lecture, the ArrayList, the List, the Stack and the Queue.

## 5.2 Exercise 1: Preliminaries

Create a project in Eclipse called Lab5. Add a class called Data as shown in Section 5.5. Add another class called Lab5. Look at the Data class, it simply provides an interface for storing two items of data, a name and an age. Create a main method within the Lab5 class and experiment with creating some instances of Data and displaying the values; an example is given below:

**public** **static** **void** main(String args[])

{

Data x = **new** Data("Fred",41);

x.Print();

}

## 5.2.1 Exercise 2: ArrayList and LinkedList

Now create an ArrayList designed to hold items of type Data.

E.g. ArrayList<Data> array = **new** ArrayList<Data>();

Add to the array list the following data using the add method:

|  |  |
| --- | --- |
| **Name** | **Age** |
| Fred | 21 |
| Jo | 43 |
| Zoe | 37 |

Add to class Lab5 the PrintCollection method from Section 5.6. Test the method on the array list that you have created. How would you add Harry aged 78 between Jo and Zoe? Display the contents of the array to see if you have done this correctly.

Replace the array list for a LinkedList. Run the code, what do you notice? Use the JavaDocs documentation to try and locate any of the differences between ArrayList and LinkedList.

## 5.2.2 Exercise 3: ArrayList Implementation Considerations

The ArrayList has some inherent behaviour that can result in some odd runtime errors.

Create an ArrayList called ArrayA containing the three items as in the table above. Then create an empty ArrayList called ArrayB.

Add and run the following code:

*PrintCollection*(ArrayA);

System.*out*.println();

ArrayB = ArrayA;

*PrintCollection*(ArrayB);

System.*out*.println();

ArrayA.remove(1);

*PrintCollection*(ArrayB);

What do you notice? How can you explain this?

Now create an additional two ArrayLists called ArrayC and ArrayD. Fill ArrayC with the three items as in the table above (as you did when creating ArrayA).

Add and run the following code. What do you notice? How can you explain this?

*PrintCollection*(ArrayC);

System.*out*.println();

ArrayD = (ArrayList<Data>)ArrayC.clone();

ArrayC.remove(1);

*PrintCollection*(ArrayC);

System.*out*.println();

*PrintCollection*(ArrayD);

System.*out*.println();

What you are seeing is the fact that the statement:

ArrayB = ArrayA;

does not copy the contents of ArrayA to ArrayB, it simply points ArrayB at the contents of ArrayA. This means that both arrays share the same data. Any items added or deleted to one will be effectively added or deleted from the other.

However the statement:

ArrayD = (ArrayList<Data>)ArrayC.clone();

copies all of the elements from ArrayC to ArrayD. Additions or deletions will only affect the individual array (and elements).

Test this out by adding and deleting elements from ArrayA and ArrayC and displaying the elements from ArrayB and ArrayD respectively.

**You are likely to encounter this problem in a later worksheet; be sure you understand what is happening and why!**

## 5.3 Exercise 3: Stacks

A stack can be created as follows:

Stack<Data> stack = **new** Stack<Data>();

You add items to a stack using the push method. You remove items by using the pop method. Add all of the items from the array list to the stack and then use the PrintCollection method to display the contents.

Implement the following code that manipulates the stack, what does it do?

**while**(stack.isEmpty() == **false**)

{

stack.pop().Print();

}

System.*out*.println(stack.size());

## 5.4 Exercise 4: Queues

A queue can be created as follows:

ArrayBlockingQueue<Data> q = **new** ArrayBlockingQueue<Data>(10);

This creates a queue based on a cyclic array as discussed in the lectures. The value of 10 is the maximum size of the queue.

You add (*enqueue*) items to a stack using the add method. You remove (*dequeue*) items by using the pop method. Add all of the items from the array list to the queue and then use the PrintCollection method to display the contents.

Implement the following code that manipulates the queue, what does it do? Compare the differences with the similar code for the stack.

**while**(q.isEmpty() == **false**)

{

q.poll().Print();

}

System.*out*.println(q.size());

Add the following code:

**for**(**int** i=0;i<10;++i)

{

q.add(**new** Data("Test:"+String.*valueOf*(i),i));

}

*PrintCollection*(q);

Run the code. Observe the results. Now change the for loop limit from 10 to 20. What happens? Why? Now change the add method to the offer method. What happens? What does offer return? Under what conditions?

## 5.5 The Data Class

This class holds two values, one for a name and one for an age. The constructor requires a name and an age. There are appropriate get and set methods, along with a print method.

**public** **class** Data

{

**private** String name;

**private** **int** age;

Data(String n,**int** a)

{

name = n;

age = a;

}

**public** String GetName()

{

**return**(name);

}

**public** **void** SetName(String n)

{

name = n;

}

**public** **int** GetAge()

{

**return**(age);

}

**public** **void** SetAge(**int** a)

{

age = a;

}

**public** **void** Print()

{

System.*out*.print(("("+GetName()));

System.*out*.print(",");

System.*out*.print(GetAge());

System.*out*.print(") ");

}

}

## 5.6 The PrintCollection Method

This method displays the contents of any object that inherits from Collection containing elements of class Data. Note that all of the items are displayed on the same line with a new line at the end.

**public** **static** **void** PrintCollection(Collection<Data> c)

{

**for** (Iterator<Data> iter = c.iterator(); iter.hasNext();)

{

Data x = (Data)iter.next();

x.Print();

}

System.*out*.println();

}