Laboratory 5 – Week 6

## Objects and Arrays

### 5.1 Introduction

**Note that: this worksheet *is* one of the worksheets from which your laboratory worksheets portfolio of work will be assessed [CodeRunner tests].**

This laboratory worksheet covers the use of **Objects** and **Arrays** in Java. 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.

### 5.2 Preliminaries

Create a project in Eclipse called CS1702\_Lab5. Add a class called Data as shown in section 5.7. We are now going to create another class within this project, our first multi-class project/program. Now add another class called CS1702\_Lab5, with an associated main method as usual.

Look at the Data class, it simply provides an interface for storing two items of data, a name and an age. Within the main method of the CS1702\_Lab5 class we will 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();

}

### 5.3 Using an ArrayList

Now create (within class CS1702\_Lab5) an ArrayList designed to hold items of type Data.

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

Remember to add the import statement from the lecture notes.

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

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

Add to class CS1702\_Lab5 the PrintDataArray method from section 5.8. 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.

### 5.4 Casting

Type in the following small amount of code:

**int** x = 10;

**double** y = x;

System.*out*.println(y);

This program is not very exciting, but we will use it to demonstrate how to convert from one data type to another. Swap the data types of variables x and y around. What happens? We should get an error; Java will tell us that it does not know how to put a double into an int. In the previous case, we should note that the set of real numbers contains the set of integers.

Modify your code as follows:

**double** x = 10.3;

**int** y = (**int**)x;

System.*out*.println(y);

This is called **casting**, the notation (<type>) instructs Java to convert (if possible) from one data type to another. Run some experiments using the example code above to determine if Java rounds up or down.

Another example of where we might need casting is as follows:

Byte by = **new** Byte(255);

This will not work.

Byte by = **new** Byte((**byte**)255);

This will.

Note that longs and floats have a shorthand notation to implicitly state a variable type:

Long v1 = 10L;

Float v2 = 10.3F;

Observe what happens when you remove the L and F. Rewrite these lines using casting.

### 5.5 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:

*PrintDataArray*(ArrayA);

System.*out*.println();

ArrayB = ArrayA;

*PrintDataArray*(ArrayB);

System.*out*.println();

ArrayA.remove(1);

*PrintDataArray*(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?

*PrintDataArray*(ArrayC);

System.*out*.println();

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

ArrayC.remove(1);

*PrintDataArray*(ArrayC);

System.*out*.println();

*PrintDataArray*(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!!!**

### 5.6 Using Arrays With Your Finch

Does your Finch go in a straight line? Does your Finch go twice as far/fast at speed 200 than at speed 100? We are going to consider and answer these questions in this part of the worksheet!

For the following speeds: 25, 50, 75, 100, 125, 150, 175, 200, 225, 250 we are going to test how far the Finch goes and how far it deviates.

Create a 10 row by 3 column integer array that holds the speeds as specified above, the first column and second column will correspond to the speeds for the left and right wheel respectively and the third column will be for the time in milliseconds (ms) the Finch travels for, initially set the value to 5000 ms. You should be able to use a for loop to populate the array. Check that you have created the array correctly by using the PrintArray method as shown in section 5.9.

Once you have entered the data correctly, use a for loop to iterate through each row, instructing the Finch robot to move according to the numbers in each row. Pause after each movement using the code in section 5.10. For the first set of experiments, see if your Finch moves in a straight line for each set of instructions. If it does not, create two double variables called LeftTrim and RightTrim and set their values initially to 1.0 each. Use these variables as multipliers to the left and right wheel movement amounts. Change them up and down by small amounts to get the Finch to travel in a straight line. Be careful not to exceed 255 for either of the wheel velocities.

Once you have your Finch travelling in an approximately straight line, run the experiments again (several times), but this time measure how far the Finch moves each time. Record these distances so that you can plot a graph in Excel of the distance travelled against the corresponding wheel velocity settings. Hopefully you should obtain a straight line!

**Make sure you save your graph for the assessment. Note that these calibrations and results will be VERY useful for your work later on in the group project.**

### 5.7 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(") ");

}

}

### 5.8 The PrintDataArray Method

This method displays the contents of an ArrayList that contains Data objects.

**private** **static** **void** PrintDataArray(ArrayList<Data> array)

{

**for**(**int** i=0;i<array.size();++i)

{

array.get(i).Print();

}

}

### 5.9 The PrintArray Method

This method displays the contents of a two dimensional array. Understand how it works since it is very useful!

**private** **static** **void** PrintArray(**int**[][] array)

{

**for**(**int** i=0;i<array.length;++i)

{

**for**(**int** j=0;j<array[i].length;++j)

{

System.*out*.print(array[i][j] + " ");

}

System.*out*.println();

}

}

### 5.10 Click OK To Continue…

To create a message box that pauses your program until the OK button is clicked, add the following to the very top of your program:

**import** javax.swing.JOptionPane;

And then use the following code where needed:

JOptionPane.*showMessageDialog*(**null**,"Click OK to continue...");

As a note, the **null** keyword is used to indicate that an object is not pointing to anything. It is rather like declaring an integer that contains zero or a string that contains no characters ("").