# assessed Lab 4

# Objects and Arrays

## ASSESSMENT INFORMATION

**This worksheet *is* one of the seven assessed lab sheets.**

**It can be assessed within the next *3* weeks. Let me know *in advance* when you’d like to be assessed.**

**Last day for assessment (deadline: Week 12 Term 1).**

**Do not forget to have it ‘signed off’ after you have been assessed.**

## Introduction

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.

## 2 Preliminaries

Create a project in Eclipse called CS1002\_Lab4.

## 3 declaring and using arrays and arraylists

### Exercise 1

Add a new class called Heights. It should have a main method, why?

Suppose there are ten students in a class and these students have varied heights. In this exercise, you need to declare and create an array (e.g., named ‘heights’) to store all of the heights of these students. Initialise the array with any different heights of your own (you can do this using the shortcut). The heights are all ‘double’ values.

1. Print the **length** of the array
2. Print all values in the array, using a loop.
3. Now calculate the average height of this class and print it out.
4. Search for the minimum height and print it. Follow the example from the lecture.
5. Now do the same with the maximum.

### Exercise 2

Use ArrayList to set up a new arraylist (e.g., named ‘heightsList”) to store the original heights of those 10 students. Do not forget to add the **import** statement in the beginning!

Note, you need to use ‘new Double(x)’ to create and store Double objects in the arraylist rather than store ‘double’ values directly. This is explained further in section 6.

So after you create the ArrayList type:

heightsList.add(**new** Double (170));

or :

heightsList.add(170.0);

and not:

heightsList.add(**170);**

1. Use the ‘**remove**’ method of ArrayList to remove the **third** element of the array.
2. Then, **add** 5 more different heights into this arraylist.
3. Do a bit of research online and find a way to manually **sort** the heights in the list in ascending order!
4. Print the length of the arraylist and all of the values in it.
5. Search for the minimum and maximum heights in the arraylist. Print the minimum and maximum values.

## 4 creating classes and objects

Add a class called Member. Copy-paste the code as shown in section 9.

We are now going to create another class within this project, our first multi-class project/program. Now add another class called CS1002\_Lab5, with an associated main method as usual. This will be the ‘controlling class’ that we will use to create objects.

### Exercise 3

Look at the Member class.

Identify and make a note of the following elements of the source code:

* *Class declaration (modifier, name)*
* *Field declaration (modifier, type, name)*
* *Constructor (name)*
* *List of getters (modifier, return type, name, parameters)*
* *List of setters (modifier, return type, name, parameters)*
* *List of other methods (modifier, return type, name, parameters)*
* ***Now add appropriate comments for each method.***

### Exercise 4

Within the main method of the CS1002\_Lab5 class we will experiment with creating some instances (objects) of Member and displaying the values of the fields; an example is given below.

Create two more objects and test all their methods.

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

{

Member member1 = **new** Member("Lela",41);

member1.printInfo();

}

## 5 Using Array and arrayList with objects

### Exercise 5

1. Now create (within class CS1002\_Lab5) an ArrayList designed to hold items of type Member. It should look like that:

ArrayList<Member> memberList = **new** ArrayList<Member>();

Remember to add the import statement before the class declaration (See the lecture notes).

1. Create three more Member objects (with the values as shown in the table below).

Add them to the array list using the add method.

|  |  |
| --- | --- |
| **Name** | **Age** |
| Leonardo | 21 |
| Michelangelo | 43 |
| Rafaelo | 37 |

1. Add to class CS1002\_Lab5 the printMemberData method from section 10.

**Important:**  Paste this code outside the brackets of the main method (but inside the class).

1. Test the method on the array list that you have created.

You do this by adding this statement inside the main method.

*printMemberData*(memberList);

Why do you think it has to be inside the main method?

1. How would you add Donatelo aged 58 between Leonardo and Michelangelo? Display the contents of the array to see if you have done this correctly.

## 6 Casting

### Exercise 6

In a new class (that has a main method), 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.

## 7 ArrayList Implementation Considerations

The ArrayList (and arrays too) has some inherent behaviour that can result in some odd runtime errors.

### Exercise 7

Create an ArrayList called mylistA containing the three member objects as you did for section 5 above. Then create an empty ArrayList called mylistB. Add and run the following code:

printMemberData(mylistA);

System.*out*.println();

mylistB = mylistA;

printMemberData(mylistB);

System.*out*.println();

mylistA.remove(1);

printMemberData(mylistB);

What do you notice? How can you explain this?

Now create an additional two ArrayLists called mylistC and mylistD. Fill mylistC with the three member objects as in section 5 above (as you did when creating mylistA).

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

printMemberData(mylistC);

System.*out*.println();

mylistD = (ArrayList<member>)mylistC.clone();

mylistC.remove(1);

printMemberData(mylistC);

System.*out*.println();

printMemberData(mylistD);

System.*out*.println();

What you are seeing is the fact that the statement:

mylistB = mylistA;

does not copy the contents of mylistA to mylistB, it simply points mylistB at the contents of mylistA. 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:

mylistD = (ArrayList<Data>)mylist.clone();

copies all of the elements from mylistC to mylistD. Additions or deletions will only affect the individual arraylist (and elements).

Test this out by adding and deleting elements from mylistA and mylistC and displaying the elements from mylistB and mylistD respectively.

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

## 8 Using Arrays With Your Finch

### Exercise 8

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.

1. Create a 3 row by 10 column integer array (call it myarray) that holds the speeds as specified above. The first row and second row will correspond to the speeds for the left and right wheel respectively and the third row will be for the time in milliseconds (ms) the Finch travels for. This value could be set to 5000 ms.

You can imagine the two-dimensional array as shown below:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 25 | 50 | 75 | 100 | 125 | 150 | 175 | 200 | 225 | 250 |
| 25 | 50 | 75 | 100 | 125 | 150 | 175 | 200 | 225 | 250 |
| 5000 | 5000 | 5000 | 5000 | 5000 | 5000 | 5000 | 5000 | 5000 | 5000 |

You should be able to use a for loop to populate the array.

1. Check that you have created the array correctly by using the PrintArray method. The code is in section 11. Like before, copy-paste this method outside the main method.

Then in order to use it type inside the main method:

*PrintArray*(myarray);

You should get something like this:

25 50 75 100 125 150 175 200 225 250

25 50 75 100 125 150 175 200 225 250

5000 5000 5000 5000 5000 5000 5000 5000 5000 5000

1. 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 12.

## 9 The Member Class

This class has two fields, one for a name and one for an age. The constructor requires a name and an age. There are appropriate getter and setter methods, along with a print method.

**public** **class** Member

{

**private** String name;

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

public Member(String n,**int** a)

{

name = n;

age = a;

}

**public** String getName()

{

**return** name;

}

**public** **void** setName(String newname)

{

name = newname;

}

**public** **int** getAge()

{

**return** age;

}

**public** **void** setAge(**int** newage)

{

age = newage;

}

**public** **void** printInfo()

{

System.*out*.println("Name: "+ getName());

System.*out*.println("Age: "+ getAge());

System.*out*.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

}

}

## 10 The printMemberData Method

This method displays the contents of an ArrayList that contains Member objects. Remember to paste it inside the class **but outside the main method.**

**private** **static** **void** printMemberData(ArrayList<Member> mylist)

{

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

{

Member person=mylist.get(i);

person.printInfo();

}

}

## 11 The PrintArray Method

This method displays the contents of a two dimensional array. Understand how it works since it is very useful! Remember to paste it inside the class **but outside the main method.**

**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();

}

}

## 12 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 ("").