MSc Software Development

CSC7051 Programming 2

Practical 5

**ArrayLists & Threading**

**Part 1**

Create a class “Student” that will hold details such as first / last name, age and a student number.

Create an ArrayList called **studentList** that allows for the adding of the new Student class. Instantiate a number of student objects and add them into the ArrayList. This studentList can be used to model a programming class of software engineers.

Create a method called printStudents, that takes as an argument the ArrayList of students and have it print out the details in a readable format. For this you may override *toString* in your student class.

e.g.:

for(Student stident : studentList){

System.out.println(student); // Will work if you have overridden toString in the Student class.

}

Use the *enhanced for* loop syntax (as shown above) but also then implement the same functionality using an Iterator.

Finally have a **print** statement that will output the number of people that are contained within the ArrayList.

**Part 2**

Create a method called ***averageAge***, which will take in the list of students and compute and output the average age of the programming class.

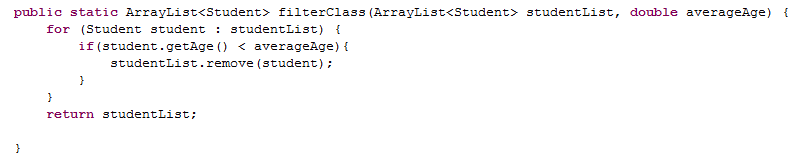
Now create a method that will take the list of students and the average age of the student list and have it filter out people who are below the average age.

e.g:

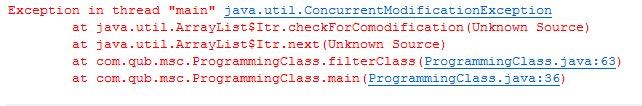
public ArrayList<Student> filterClass(ArrayList<Student> studentList, double averageAge)

**NOTE**

When in a ***for*** loop / ***enhanced for*** loop you cannot remove items from a collection. So in our case we cannot just remove the element from the ArrayList.



This seems valid, but will result in this:



There are two ways we can deal with this problem. The first is to create another **ArrayList** within your method that will represent the students that are indeed above this average age threshold. Therefore when we loop through the original studentList passed in via the method, if the student is older than the average age, we add them into the new ArrayList. This is then returned from the method. Try to implement this solution.

The alternative way is similar to how we use the Iterator technique for cycling through the ArrayList. However, instead of called studentList.remove(student), we call the remove method of the iterator.

***e.g. Iterator<Student> it = studentList.iterator();***

***// cycle through the elements of the studentList, and test if they are below the age boundary***

***it.remove();***

Again try to implement this solution.

**Part 3**

Create two ArrayLists that will represent two classes in a university year group.

i.e.

ArrayList<Student> programmingClass = new ArrayList<Student>()

ArrayList<Student> databaseClass = new ArrayList<Student>();

Now create some students objects and add these to the ***programmingClass***.

Create some more student objects and add these to the ***databaseClass***.

Finally create some more student objects and add these to **BOTH** classes.

e.g.

Student s1 = new Student();

Student s2 = new Student();

programmingClass.add(s1);

databaseClass.add(s1);

programmingClass.add(s2);

databaseClass.add(s2);

Create a method that will take both of these lists and return a new list that is the merge of the inputs. The resulting list however should have no duplicates. The method signature should be:

**private** **static** ArrayList<Student> merge(ArrayList<Student> programmingList,

ArrayList<Student> databaseList) {

Finally create a method that will create a class report, which is the result of the merging of these two classes. Output this to a file called ***class\_report.txt***.

**public** **void** writeFile(List<Student> students) {

**try** {

File output = **new** File(“class\_report.txt”);

**if** (!output.exists()) {

output.createNewFile();

}

FileWriter fw = **new** FileWriter(output);

BufferedWriter bw = **new** BufferedWriter(fw);

//format the student details and write to the file here.

bw.close();

} **catch** (IOException e) {

System.*out*.println(e.getMessage());

}

}

**Threads**

**Exercise 1**

1. Create a class that within a **main** method starts another **Thread** (countdown) that counts down from 10000 to 1 and outputs the ***count*** to screen.
2. Add a delay to the **countdown** (0.1 second) between number decrements.
3. After the **countdown** thread starts add a delay to the **main** method of **2** seconds.
4. When the **main** Thread awakens send an interrupt to the **countdown** thread to cause it to stop.
5. Configure the **main** Thread to accept a command line argument to set the length of delay in step c.

**Exercise 2**

1. Create a class that instantiates an ArrayList of type Integer (declare it statically)
2. Populate the ArrayList with the values 1 – 10 inclusive.
3. Create a Threaded class *ControlQueue* (implements runnable) that will continually remove the first item in the ArrayList until there are no further items. Each time an item is removed the array contents should be displayed on screen.
4. Run the main method to complete steps a, b and then start the Thread *ControlQueue*.
5. Add a delay of 2 seconds between removing items in *ControlQueue*.
6. Run the program again.
7. Alter the main method to enable another population of the ArrayList. After the *ControlQueue* Thread is started (step d) then add the numbers 11-20 (inclusive) to the ArrayList.
8. Run the program again.
9. Add a delay to step g – each item (11-20) should be added 0.5 seconds after each other.
10. Run the program again.
11. For consideration – is there a potential Race Condition? Should any areas of the code be synchronised?