Computing and Information Systems/Creative Computing
University of London International Programmes
CO1109 Introduction to Java and object-oriented programming

# Coursework assignment 2 2016–2017

## Introduction

This is Coursework assignment 2 (of two coursework assignments in total) for 2016–2017. The assignment asks that you demonstrate an understanding of static and instance methods including *toString()* methods, sorting and searching, reading and writing to a file, defining classes and exception handling. This assignment also introduces the *ArrayList* class.

**Electronic files you should have:**

- *Student.java*
- *ProcessModuleMarks.java*
- *marks.txt.*

**What you should hand in: very important**
There is one mark allocated for handing in uncompressed files – that is, students who hand in zipped or .tar files or any other form of compressed files can only score 49/50 marks.

There is also another mark allocated for handing in just the .java files asked for without putting them in a directory; students who upload their files in directories can only achieve 49/50 marks.

At the end of the questions, there is a list of files to be handed in – **please note the hand-in requirements supersede the generic University of London instructions**. Please make sure that you hand in the required **electronic versions** of your .java files. Failure to do so will result in 0 marks. Class files are **not** needed, and any student giving in only a class file will not receive any marks for that part of the coursework assignment, **so please be careful about what you upload as you could fail if you submit incorrectly**.

**Programs that do not compile will not receive any marks.**

**The examiner intends to compile and run your Java programs; for this reason, students who hand in files containing their Java classes that cannot be compiled (*e.g.* PDFs) will not be given any marks for that part of the assignment.**

Please put your name and student number as a comment at the top of each .java file that you hand in.

**CO1109 Coursework assignment 2**

Consider the *Student* class, below:

```
public class Student{

     private String name;
     private int exam;
     private int cwk;
     private int total;
     private String grade;

     public Student(String name, int exam, int cwk){
          this.name=name;
          this.exam=exam;
          this.cwk=cwk;
          this.total=calcTotal();
          this.grade=calcGrade();
     }

     private int calcTotal(){
          return (int)Math.round(0.6*exam+(1.0-0.6)*cwk);
     }

     private String calcGrade(){
          if (exam<35||cwk<35) return ("F");
          //if either exam or cwk component less than 35 fail,
          even if total mark is 40 or greater
          if (total<40) return ("F");  //the pass mark is 40
          if (total<50) return ("D");
          if (total<60) return ("C");
          if (total<70) return ("B");
          return ("A");
     }

     public int getTotal(){
          return total;
     }

     public String toString(){
          return name + " " + " " + exam + " " + cwk  + " " +
             total + " " + grade;
     }
}
```

You will note that the *Student* class has five instance variables: *name, exam, cwk, total* and *grade.* Constructors exist to initialise the instance variables of an object and make it ready for use. In this case, the constructor takes three of the values of the instance variables from the user, and then calls methods to work out the value of the *total* and *grade* instance variables. The methods to calculate *total* and *grade* (*calcTotal()* and *calcGrade()*) are instance methods – you can tell this because they do not have 'static' in their heading. Since these methods are setting the value of instance variables, it is good practice to make them instance methods.

You will also note that the five instance variables have private access; this means they can only be accessed from within their own class. This is also good practice. If access is needed to instance variables this is normally granted through public instance methods called setters and getters – this is to protect the fields of the object from unexpected updates. Getters are so called because they get the value of the instance variable and return it. The *Student* class has one getter, *getTotal(),* that returns the value of the *total* variable. Note that calling the method *getTotal()* follows a Java naming convention for getters, *i.e*. the first part of the name is 'get', and the second part is the name of the variable the method is getting; this should startwith a capital letter. Setters allow the value of an instance variable to be changed; the *Student* class needs no setters.

The *Student* class has been adapted from the one found on pages 104 and 105, in Chapter 12 of Volume 2 of the subject guide. This chapter is concerned with the *Vector* class and its similarities to the *Array* class. *Vectors* are more flexible than *Arrays* because they can grow and shrink dynamically as your program requires them to. However, *Vectors*, while their use is not deprecated, are not really used by developers anymore; instead, an *ArrayList* is more likely to be used. Like *Vectors*, *ArrayLists* are considered by Java to be collections, and both classes implement the *List* interface. [Chapter 8 of Volume 2 of the subject guide covers arrays] In Chapter 12 of Volume 2 of the subject guide, pages 95 and 96 list five methods of the *Vector* class that have equivalent methods in the *ArrayList* class as follows.:

| **Vector (from subject guide)** | **ArrayList** |
|---|---|
| 1. `void addElement()`<br>Adds an Object to the end of a Vector | `boolean add(E e)`<br>Adds the specified element to the end of the ArrayList. |
| 2. `Object elementAt(int i)`<br>Returns the ith element | `get(int i)`<br>Returns the ith element. |
| 3. `int Size()`<br>Returns the number of elements | `int size()` Returns the number of elements. |
| 4. void `removeElementAt(int i)`<br>Removes the ith element | `remove (int i)`<br>Removes the ith element. Shifts any subsequent elements to the left (subtracts one from their indices). |
| 5. `void setElementAt(Object x, int i)`<br>Changes the ith element to x. | `add(int i, E element)`<br>Adds an element at the specified position. Shifts the element currently at that position (if any) and any subsequent elements to the right (adds one to their indices). |

You can find more information in the API, see later: *Reading for the assignment*.

Some of the tasks in this coursework assignment are based on exercises using *Vectors* in Chapter 12 of Volume 2 of the subject guide. Instead of *Vectors*, this assignment specifies the use of *ArrayLists*. You have been given the *ProcessStudentMarks* class, and you are required to write some static methods to manipulate an *ArrayList* of type *Student* (note that the type of variable held by an *ArrayList* is denoted with angled brackets, *e.g.*

*ArrayList<Student>*).

Please note that the main method of the *ProcessStudentMarks* class has been written to test the methods. **Please do not change the main method as it will be used, exactly as given, by the examiner to test your class.** Instead, you should make sure to write methods that will make the statements in the main method legal. Once you have written a method in the *ProcessStudentMarks* class, remove the comment marks in front of the method call in the main method, re-compile and run the class to test your new method.

Please complete the following tasks:

1. Write getter methods for the remaining four instance variables in the class *Student.* In your answer, be sure to follow the Java naming convention for getters. [4 marks]

2. The *StudentsInList_2(String)* method, calls another method, *positionOfNext(ArrayList<Student>, int)*, to determine where to place each newly created *Student* object in the *ArrayList*. This is done in order to produce an *ArrayList* of *Student* sorted by the total student mark in ascending order. The total (or final) module mark is given by the *total* instance variable in the *Student* class.

   Write the body of the *positionOfNext(ArrayList<Student>, int)* method (you will find a skeleton method in the *ProcessModuleMarks* class). You will need to think carefully about the logic of the method, and remember that it is possible to write a method that is syntactically correct (*i.e*. it will compile) but logically wrong such that your program and/or method does not do what you expect it to do. [6 marks]

3. Write the *DisplayStudents(ArrayList<Student>)* method. This method displays the entire contents of a given *ArrayList<Student>.* It uses the *Student* class's *toString()* method to display each element of the *ArrayList* on its own line. [5 marks]

4. Write the *findStudentName(ArrayList<Student>, String)* method. This method will find if a *Student name* variable starts with the given *String*, and it will print out **all** the matches that it finds to standard output. Note that the method should be case insensitive, *e.g.* if given "Ace", "ACE", or "ace" it will find and print the four *Student* objects whose *name* field starts with "Ace". You may wish to consult the *String* API for *String* methods to help you complete this part of the assignment, see below: *Reading for the assignment.* [6 marks]

5. The *StudentsInList(String)* and the *StudentsInList_2(String)* methods both throw, and do not handle, the potential *FileNotFoundException*. Rewrite both methods to handle the potential *FileNotFoundException* with try/catch. [6 marks]

6. Write the *findStudentGrade(ArrayList<Student>, String)* method. This method will find **all** students with a particular grade (given by the *String* parameter) and print them to standard output. [5 marks]

7. Write a method to implement binary search on the sorted *ArrayList*, *students2*. Your method should have the heading:
*public static boolean binarySearch(ArrayList<Student> a, int thing)*
and will return *true* if it can find an element in the *ArrayList* with a total mark given by the *int* parameter, and *false* if it cannot find such an element. [6 marks]

8. Write the *StudentsToFile(ArrayList<Student>, String)* method. Given an *ArrayList* of type *Student* and a *String* file name, this method will write the members of the *ArrayList* to a text file with the given name, with one element to a line, using *Student's toString()* method. Make sure to handle any possible exception with try/catch. [6 marks]

9. You may note that the *toString()* method of the *Student* class does not label the fields when it prints them. If it did, it could make the output of the *ProcessModuleMarks* class more readable. Rewrite the *Student* class's *toString()* method to label the fields of the output. [4 marks]

**Reading for the assignment**

The Java API particularly:

- https://docs.oracle.com/javase/7/docs/api/java/util/ArrayList.html
- https://docs.oracle.com/javase/7/docs/api/java/lang/String.html

Volume 2 of the CO11109 subject guide, and in particular:

- Chapter 7, *Files and Streams,* sections 7.1 to 7.6.1 inclusive.
- Section 8.6 (*Searching*); 8.6.1 (*Linear Searching*); 8.6.2 (*Binary Searching*).
- Chapter 9, *Defining Classes* (please read the whole chapter).
- Section 10.6, *More about instance methods.*
- Chapter 11, *Exception Handling* (please read the whole chapter).
- Sections 12.8, 12.9, 12.10.

**Deliverables**

- An electronic copy of your revised: *Student.java.*
- An electronic copy of your revised: *ProcessModuleMarks.java.*

## MARKS FOR CO1109 COURSEWORK ASSIGNMENT 2

The marks for Coursework assignment 2 are indicated at the end of each question in [.] brackets and add up to 48. There are another two marks available for handing in **uncompressed** .Java files and files that are **not** contained in a directory. This amounts to 50 marks altogether. There are another 50 marks available from Coursework assignment 1.

| | |
|---|---|
| Total marks | [48 marks] |
| Mark for giving in uncompressed files | [1 mark] |
| Mark for giving in standalone files; namely, files **not** enclosed in a directory | [1 mark] |
| **Total marks for Coursework assignment 2** | **[50 marks]** |

**[END OF COURSEWORK ASSIGNMENT 2]**