University of London International Programmes

Computing and Information Systems/Creative Computing

CO1109 Introduction to Java and object-oriented programming

Coursework assignment 2 2017–18

Introduction

This is Coursework assignment 2 (of two coursework assignments) for CO1109 for 2017–18. The assignment asks that you demonstrate an understanding of static and instance methods including *toString()* methods; sorting and searching; reading and writing to files; constructors and exception handling. This coursework assignment also introduces the *ArrayList* class.

Files you should have for Part A:

- *DictionaryMethods.java*
- *smallDictionary.txt*
- *vSmallDictionary.txt*

Files you should have for Part B:

- *Finalist.java*
- *FinalistComparator.java*
- *ProcessDegreeMarks.java*
- *finalMark.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 one 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 each section there is a list of files to be handed in – **please note these hand-in requirements supersede the generic University of London instructions**. Please make sure that you give in **electronic versions** of your .java files since you cannot gain any marks without handing in the **.java** files asked for. 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 examiners will compile and run your Java programs; students who hand in files containing their Java classes that cannot be compiled (*e.g.* PDFs) will not receive 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.

Part A: *DictionaryMethods.java*

Notes on `ArrayList`
https://docs.oracle.com/javase/7/docs/api/java/util/ArrayList.html

Chapter 12 of the subject guide, Volume 2, concentrates on the `Vector` class, and its similarities to arrays. Vectors are more flexible than arrays, because they can grow and shrink dynamically, as your program requires. While the `Vector` class is not deprecated, it has been replaced by the `ArrayList` class; an `ArrayList` can also grow and shrink dynamically. Both classes implement the `List` interface.

Arraylists are considered by Java to be a collection. `Collections` is a class that has useful static methods that operate on any class that Java considers to be a collection; see https://docs.oracle.com/javase/7/docs/api/java/util/Collections.html. This means that, among other things, we can sort an `ArrayList` using *Collections.sort()*[*].

Pages 95 and 96 of Volume 2 of the subject guide list five methods of the `Vector` class, which have equivalent methods in the `ArrayList` class as follows.

`Vector` *Method numbering from subject guide*	**`ArrayList`**
1. `void addElement()` Adds an Object to the end of a `Vector.`	`boolean add(E e)` Adds the specified element to the end of the `ArrayList.`
2. `Object elementAt(int i)` Returns the *i*th element.	`get(int i)` Returns the *i*th element.
3. `int Size()` Returns the number of elements.	`int size()` Returns the number of elements.
4. void `removeElementAt(int i)` Removes the *i*th element.	`remove (int i)` Removes the *i*th element. Shifts any subsequent elements to the left (subtracts one from their indices).
5. `void setElementAt(Object x, int i)` Changes the *i*th element to x.	`add(int i, E element)` 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).

[*] See the *sortDictionary()* method of the *DictionaryMethods* class

Consider the *DictionaryMethods* class. When the program starts, an `ArrayList` called *dictionary* is filled with words from the file *smallDictionary.txt*.

Compile and run the program. You should see this menu:

```
Choose one of the following options:

1) check that a word is in the dictionary
2) show all words in dictionary
3) show all words in dictionary containing a search String
4) show all words in dictionary starting with a search String
5) show all words in dictionary ending with a search String
6) show all words in dictionary of a certain length
7) Quit

Enter your choice:
```

If the user enters a `String` that cannot be parsed to one of the numbers 1–7, then the program returns: `Unknown option`. For four of the valid menu choices the program will return “I don’t know how to do that”, since only options 1, 2 and 7 are implemented.

Note that all of the methods of the *DictionaryMethods* class are instance methods. The class has a constructor, which initialises its four instances variables, and then runs the *startLoop()* method. The *startLoop()* method runs the user interaction loop, effectively running the program. This means that to run the class we only need to make an instance of a *DictionaryMethods* object, which is done in the main method.

1. Option 1 checks if the `String` the user enters is in the dictionary. If the String is in the dictionary the user is told:
   `Word entered is in the dictionary`
   If the `String` is not in the dictionary there is no output from the method.

   Amend the method to do the following:
   - If the search `String` is found the message printed is:
     `<word> is in the dictionary`
   - If the search `String` is not found, the message is:
     `<word> is not in the dictionary`

   For example, if the `String` entered is “klajsd” then the message output will be:
   `klajsd is not in the dictionary` **[2 marks]**

2. Option 2 displays all of the words that have been read into the *dictionary* `ArrayList` variable by the *readDictionary(String)* method. Test the method. You should find that it is quite slow, taking a few seconds to display the *dictionary*.

   Rewrite the body of the method so that it works faster. **[2 marks]**

3. Write the following methods.

- *private void contains();* (menu choice 3)
- *private void startsWith();* (menu choice 4)
- *private void endsWith();* (menu choice 5)
- *private void thisLong();* (menu choice 6)

Make sure that each of your methods outputs an appropriate message to the user, whether or not the method finds anything in the *dictionary* to output.

Once you have written a method, add it to the switch block so that when the user chooses the corresponding option from the menu, the method runs.

You can tell what the methods are supposed to do by the messages displayed in the menu, and by the example output in Appendix 1. **[12 marks]**

4. There is a method called *changeDictionary()* that is not included in the switch block or the menu. Add the method as choice 7 to the switch block, and make all necessary changes so that menu choice 7 runs the method, and menu choice 8 quits the program. **[2 marks]**

5. The *changeDictionary()* method is supposed to replace the current dictionary, with the contents of a file whose name is entered by the user. Test the *changeDictionary()* method, changing the file to *vSmallDictionary.txt*, and choosing option 2 to display the new dictionary. You should find that the method does not work as intended. Identify and correct the error. **[2 marks]**

6. There is another issue with the *changeDictionary()* method. If the user enters a file name that does not exist, then the program stops with a `NullPointerException`. Amend the program so that if the user enters an invalid file name, the program enters a loop asking for a valid file name. Once the user enters a valid file name the method executes, and displays a message to the user, telling them that the dictionary has been changed. **[3 marks]**

Reading for Part A

Question 1
Volume 1 of the subject guide: Sections 4.11.4 and 4.11.5 (`String` and `int` concatenation).

Question 2
https://stackoverflow.com/questions/15177987/is-string-concatenaion-really-that-slow
You can also search the Internet or otherwise research known issues with `String` concatenation.

Questions 3 and 5
Read about `ArrayList` and `String` methods on the Java API

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

Question 6
Volume 2 of the subject guide: Chapter 11, *Exception Handling*, particularly section 11.4.

Deliverable for Part A

- An electronic copy of your revised: *DictionaryMethods.java*

Part B

```
public class Finalist{

    private String id;
    private double degreeMark;
    private String degreeClass;
    private boolean borderline;

    public Finalist(String id, double degreeMark) {
        this.id = id;
        this.degreeMark = degreeMark;
        borderline = calcBorderline();
        degreeClass = setDegreeClass();
    }

    private String assignDegreeClass(){//change method name
         if (degreeMark<40) return "FAIL";
        if (degreeMark<50) return "THIRD";
        if (degreeMark<60) return "LOWER_SECOND";
        if (degreeMark<70) return "UPPER_SECOND";
        return "FIRST";
    }

     private boolean calcBorderline(){
           double x;
           if (degreeMark<40){
                 x = 40.0-degreeMark;
                 if (x < 1.0) return true;
           }
           if (degreeMark<50){
                 x = 50.0-degreeMark;
                 if (x < 1.0) return true;
           }
           if (degreeMark<60){
                 x = 60.0-degreeMark;
                 if (x < 1.0) return true;
           }
           if (degreeMark<70){
                 x = 70.0-degreeMark;
           if (x < 1.0) return true;
           }
           return false;
    }

      public String getId(){
           return id;
      }

     public String toString() {
       String s = "ID: " + id + ", Final Mark: " + degreeMark + ",
           Classification: " + degreeClass + System.lineSeparator();
       return s;
           }
      }
```

The *Finalist* class

Consider the *Finalist* class above.

You will note that the *Finalist* class has four instance variables: *id, degreeMark, degreeClass* and *borderline.* Constructors exist to initialise the instance variables of an object, and make it ready for use. In this case, the constructor takes two of the values of the instance variables from the user, and then calls methods to work out the value of the *degreeClass* and *borderline* instance variables. The methods to calculate *degreeClass* and *borderline* (*assignDegreeClass()* and *calcBorderline())* 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 four instance variables have private access, meaning 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 'information hiding' 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 *Finalist* class has one getter, *getId(),* that returns the value of the `String` *Id* variable. Note that calling the method *getId()* follows a Java naming convention for getters*;* namely, the first part of the name is 'get' and the second part is the name of the variable the method is getting, starting with a capital letter. Setters allow the value of an instance variable to be changed; the *Finalist* class needs no setters.

In the *toString()* method, instead of "\n" as the new line character, the class uses *lineSeparator()* from the `System` class. This method returns the appropriate new line code for the system that the containing class is being run on. It is a good idea to use *lineSeparator()* as Windows and Linux have different new line codes. See
https://docs.oracle.com/javase/8/docs/api/java/lang/System.html#lineSeparator--
for more information on the `System` class.

The *ProcessDegreeMark* class
You should have a copy of the *ProcessDegreeMark* class. The class has some static utility methods for manipulating arraylists containing *Finalist* objects.

The *finalistsToFile(ArrayList<Finalist>, String)* method of the *ProcessDegreeMark* class, uses a `Comparator` to sort an `ArrayList` called *finalists* in descending order of *degreeMark*. We need a `Comparator` to do the sorting, because we cannot use *Collections.sort()* since the `ArrayList` contains an object with four fields; the method would not know which field to sort by. Writing a `Comparator` allows us to specify the field to sort by. You can read more about the `Comparator` interface here:
https://docs.oracle.com/javase/7/docs/api/java/util/Comparator.html

Please note that the main method of the *ProcessDegreeMark* class has been written to test the class. **Please do not change the main method as it will be used, exactly as given, by the examiners to test your work.** Statements in the main method that do not currently compile have been commented. You should remove the comment marks when appropriate, and test your class.

You can find example output of the test statements with all methods working as they should, in Appendix 2.

1. Write getter methods for the remaining three instance variables in the class *Finalist.* In your answer, be sure to follow the Java naming convention for getters.

   Note that the *FinalistComparator* class will not compile until one of these methods has been successfully written and the *Finalist* class compiled. **[2 marks]**

2. The *toString()* method of the *Finalist* class does not return the value of the *borderline* variable. Change the *toString()* method so that if the *borderline* variable is `true`, the method prints a second line stating "Candidate is BORDERLINE". If the *borderline* variable is false, only *s* is output, *i.e.* the output remains as it currently is. **[2 marks]**

   Examples of output from your amended *toString()* method:

```
ID: 77056, Final Mark: 72.98, Classification: FIRST

ID: 85011, Final Mark: 69.2, Classification: UPPER_SECOND
Candidate is BORDERLINE
```

3. The *findFinalistID(ArrayList<Finalist>, String)* prints to standard output the member of the *finalists* arraylist with a particular *id* number. If the method does not find an element of the arraylist with that particular *id*, nothing is output. Change the method so that if the search fails, the user is told: `No candidate found with ID number <id>` **[1 mark]**

4. Write a method, *findFinalistClass(ArrayList<Finalist>, String),* that finds and prints to standard output all members of the *finalists* `ArrayList` with a particular class. If no finalist with a particular class is found, it prints a message to the user: "No candidates found with degree class <CLASS>". For example, with the test data given, if searching for a finalist with a THIRD, the method would return:
   `No candidate found with degree class THIRD` **[5 marks]**

5. The *finalistsInList(String)* method throws and does not handle exceptions. Remove `throws Exception` from the method heading and add exception handling to the body of the method.

   [2 marks]

6. The *finalistsToFile(ArrayList<Finalist>, String)* does two things:

   - It copies its `ArrayList` parameter, and sorts the copied `ArrayList` by descending order of *degreeMark* using the *FinalistComparator*.
   - It prints the sorted `ArrayList` to a file.

   It is not good object-oriented practice for a method to do two things. Methods are more likely to be easy to reuse if they have just one very specific task.

Split the *finalistsToFile(ArrayList<Finalist>, String)* method into two methods as follows:

public static ArrayList<Finalist> sortDegreeMark(ArrayList<Finalist> a)
```
/*This method takes an arraylist parameter, and
returns a second arraylist, sorted by descending
order of the degreeMark variable.*/
```

public static void finalistsToFile2(ArrayList<Finalist> finalists, String s)
```
/*This method takes an arraylist and a String as
parameters. It saves the arraylist into a file with
the name given by the String parameter. The arraylist
is printed using the toString() method of the
Finalist class. The method handles the potential
FileNotFoundException*/
```
[6 marks]

7. Write a method *findAndSaveFinalistClass(ArrayList<Finalist>, String).* The method, similar to *findFinalistClass(ArrayList<Finalist>, String),* searches for all finalists with a particular class of degree, displays any results, or tells the user there is nothing to display as appropriate. In addition:
    - The method adds any *finalists* that meet the search criteria to a new `ArrayList`.
    - The method uses the new *finalistsToFile2(ArrayList<Finalist>, String)* method to save the new `ArrayList` into a file.
    - The method will not attempt to display an empty `ArrayList`, or to save an empty `ArrayList` into a file.
    - The file name uses the `String` parameter as part of the title, *e.g. FIRSTOnly.txt*, or *FIRSTfinalists.txt* **[5 marks]**

8. Since the *ProcessDegreeMark* class has only static methods, it would not be appropriate to make an instance of it. Write a constructor that will prevent the class from being instantiated. **[2 marks]**

Reading for Part B

General reading
Volume 2 of the subject guide, and in particular: Chapter 9, *Defining Classes* (please read the whole chapter).

Question 1
Volume 2 of the subject guide: Sections 10.6 (Instance methods).

Question 3
`String` methods on the Java API
https://docs.oracle.com/javase/7/docs/api/java/lang/String.html

Questions 4 and 5
- Volume 2 of the subject guide: Chapter 11, *Exception Handling*, particularly section 11.4
- Volume 2 of the subject guide: Chapter 7, *Files and Streams,* sections 7.1 to 7.6.1 inclusive (but ignore section 7.5.2)

Question 7
http://www.javapractices.com/topic/TopicAction.do?Id=40

Deliverables for Part B
- An electronic copy of your revised: *Finalist.java*
- An electronic copy of your revised: *ProcessDegreeMark.java*

Appendix 1

Example output of the completed *DictionaryMethods* class

```
Choose one of the following options:

1) check that a word is in the dictionary
2) show all words in dictionary
3) show all words in dictionary containing a search String
4) show all words in dictionary starting with a search String
5) show all words in dictionary ending with a search String
6) show all words in dictionary of a certain length
7) change the dictionary
8) Quit

Enter your choice: 1

This method checks if your word is in the dictionary
Enter your word
hello

hello is in the dictionary
```

```
Choose one of the following options:

1) check that a word is in the dictionary
2) show all words in dictionary
3) show all words in dictionary containing a search String
4) show all words in dictionary starting with a search String
5) show all words in dictionary ending with a search String
6) show all words in dictionary of a certain length
7) change the dictionary
8) Quit

Enter your choice: 3

This method outputs words that contain your search String
Enter your String
quez

quezon
vasquez
velasquez
```

```
Choose one of the following options:

1) check that a word is in the dictionary
2) show all words in dictionary
3) show all words in dictionary containing a search String
4) show all words in dictionary starting with a search String
5) show all words in dictionary ending with a search String
6) show all words in dictionary of a certain length
7) change the dictionary
8) Quit

Enter your choice: 4

This method outputs words that start with your search String
Enter your String
uquq

Nothing found starting with uquq
```

```
Choose one of the following options:

1) check that a word is in the dictionary
2) show all words in dictionary
3) show all words in dictionary containing a search String
4) show all words in dictionary starting with a search String
5) show all words in dictionary ending with a search String
6) show all words in dictionary of a certain length
7) change the dictionary
8) Quit

Enter your choice: 5

This method outputs words that end with your search String
Enter your String
tyledon

cotyledon
monocotyledon
```

```
Choose one of the following options:

1) check that a word is in the dictionary
2) show all words in dictionary
3) show all words in dictionary containing a search String
4) show all words in dictionary starting with a search String
5) show all words in dictionary ending with a search String
6) show all words in dictionary of a certain length
7) change the dictionary
8) Quit

Enter your choice: 6

This method outputs words of a certain length
Enter the length of the word
19

anthropomorphically
incomprehensibility
straightforwardness
```

```
Choose one of the following options:

1) check that a word is in the dictionary
2) show all words in dictionary
3) show all words in dictionary containing a search String
4) show all words in dictionary starting with a search String
5) show all words in dictionary ending with a search String
6) show all words in dictionary of a certain length
7) change the dictionary
8) Quit

Enter your choice: 7

This method changes the file used to load the dictionary
Enter the filename
small.txt
small.txt file does not exist! Please re-enter:
vSmallDictionary
vSmallDictionary file does not exist! Please re-enter:
vSmallDictionary.txt

Dictionary has been changed
Choose option 2 to view the new dictionary
Choose one of the following options:

1) check that a word is in the dictionary
2) show all words in dictionary
3) show all words in dictionary containing a search String
4) show all words in dictionary starting with a search String
5) show all words in dictionary ending with a search String
6) show all words in dictionary of a certain length
7) change the dictionary
8) Quit

Enter your choice: 2

abracadabra
abstract
boom!
chips
class
defenestrate
extend
hello
implements
instance
interface
magnificent
method
object
object-oriented
pasta
pizza
quick
shazam
static
zip
```

```
Choose one of the following options:

1) check that a word is in the dictionary
2) show all words in dictionary
3) show all words in dictionary containing a search String
4) show all words in dictionary starting with a search String
5) show all words in dictionary ending with a search String
6) show all words in dictionary of a certain length
7) change the dictionary
8) Quit

Enter your choice: 8

Bye.
Press any key to continue . . .
```

Appendix 2

Output of the test statements *DictionaryMethods* class with successfully completed methods.

```
/****************************************************/
/*******finalistsInList with invalid file name*******/

file ***.txt not found

/****************************************************/
/********finalistsInList with valid file name********/
/********display to check arraylist populated********/

ID: 62138, Final Mark: 59.9, Classification: LOWER_SECOND
Candidate is BORDERLINE

ID: 74927, Final Mark: 67.58, Classification: UPPER_SECOND

ID: 77056, Final Mark: 72.98, Classification: FIRST

ID: 85921, Final Mark: 86.83, Classification: FIRST

ID: 85011, Final Mark: 69.2, Classification: UPPER_SECOND
Candidate is BORDERLINE

ID: 75021, Final Mark: 60.63, Classification: UPPER_SECOND

ID: 84321, Final Mark: 57.35, Classification: LOWER_SECOND

ID: 65734, Final Mark: 75.71, Classification: FIRST


/****************************************************/
/*testing findFinalistID with valid and invalid data*/

ID: 75021, Final Mark: 60.63, Classification: UPPER_SECOND

No candidate found with ID number 21050

/****************************************************/
/*test findFinalistClass with valid and invalid data*/

ID: 77056, Final Mark: 72.98, Classification: FIRST

ID: 85921, Final Mark: 86.83, Classification: FIRST

ID: 65734, Final Mark: 75.71, Classification: FIRST

No candidate found with degree class THIRD

/****************************************************/
/*****run sortedFinalists then test with display*****/

ID: 85921, Final Mark: 86.83, Classification: FIRST
```

```
ID: 65734, Final Mark: 75.71, Classification: FIRST

ID: 77056, Final Mark: 72.98, Classification: FIRST

ID: 85011, Final Mark: 69.2, Classification: UPPER_SECOND
Candidate is BORDERLINE

ID: 74927, Final Mark: 67.58, Classification: UPPER_SECOND

ID: 75021, Final Mark: 60.63, Classification: UPPER_SECOND

ID: 62138, Final Mark: 59.9, Classification: LOWER_SECOND
Candidate is BORDERLINE

ID: 84321, Final Mark: 57.35, Classification: LOWER_SECOND


/***************************************************/
/*****test finalistsToFile2 with sorted arraylist*****/
/**************check file testSorted.txt**************/


/***************************************************/
/*test findAndSaveFinalistClass with valid and invalid data*/

ID: 77056, Final Mark: 72.98, Classification: FIRST

ID: 85921, Final Mark: 86.83, Classification: FIRST

ID: 65734, Final Mark: 75.71, Classification: FIRST

No candidate found with degree class THRID

/*********************THE END************************/
```

Marks for CO1109 Coursework assignment 2

The marks for each section of Coursework assignment 2 are clearly displayed against each question and add up to 48. There are another two marks available for giving in uncompressed .java files and for giving in 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 for Part A	[23 marks]
Total marks for Part B	[25 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]