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

Coursework assignment 2 2015–16

Introduction
This is Coursework assignment 2 (of two coursework assignments total) for 2015–16. The coursework assignment asks that, in part a., you demonstrate an understanding of general programming including loops and complex loops, user input and `String` methods. Part b. covers arraylists, the `String` class and its methods, and writing classes with methods and constructors.

Electronic files you should have:

Part a.
- *StringInput.java*

Part b.
- *ReceiptBefore.java*
- *receipt info.txt*

Deliverables – 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 wrapping them up in a directory structure – namely, students who upload their 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 the 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**.

Note on `ArrayList`
https://docs.oracle.com/javase/7/docs/api/java/util/ArrayList.html
The CO1109 subject guide, Volume 2, concentrates on the `Vector` class. While `Vector` is not deprecated, the class is generally no longer used. Now developers are more likely to use `ArrayList`, which has replaced `Vector`. In this coursework assignment the `ArrayList` class is used instead of `Vector`, as students will find it more useful to learn about arraylists. There are enough similarities between them that most or all of the programs in Chapter 12 of Volume 2 of the subject guide would work the same if an `ArrayList` was substituted wherever there is a `Vector`.

CO1109 Coursework assignment 2

PART A

Consider the *StringInput* class. Compile and run the program. The program will ask you for a word. Once you have entered a word, for example 'Hello', you will be presented with a menu:

```
Current word: 'Hello'
Choose one of the following options:
1)   Repeat
2)   Reverse
3)   Arrange Alphabetically
4)   Randomise
5)   Frequency Count
6)   Remove Vowels
7)   Insert Spaces
8)   Quit
Enter your choice:
```

If you then choose an option, say 7, you will see the following output:

```
H e l l o

Bye.

Tool completed successfully
```

For some of the menu choices the program will return "I don't know how to do that." In that case it will be your job to implement the associated method so that the menu choice works. Other menu choices have been implemented, but may need some fixing in order to do what they are intended to do.

1. Write the *reverse()* method. Add it to the switch block so that when the user chooses option 2 from the menu, their word is output reversed (e.g. 'Hello' becomes 'olleH'). [3 marks]

2. Run the program, enter the word 'Extend' (**note**: first letter capitalised), choose option 3 to arrange the word alphabetically. You should see the following output: `Edentx`
   If you enter 'extend' without the first letter capitalised, you should see the following: `deentx`
   You will note that the program is treating upper and lower case letters as if they are different. Fix the *alphabetise()* method so that if 'Extend' is entered the output will be: `dEentx`; namely, so that the method treats upper and lower case versions of the same letter as the same character. [3 marks]

3. Write the *randomise()* method. Add it to the switch block so that when the user chooses option 4 from the menu, their word is output with the characters in random order. [3 marks]

4. Run the program, enter a word and choose option 5. You should find that there is no output although the *frequency()* method has been written. Find and fix the problem with the method. When you have done so put a comment at the start or end of the method explaining what the problem was and how you have fixed it. Your comment should not be more than 8 sentences long. [3 marks]

5. Write the method to remove the vowels from a word. Vowels are the letters ‘a’, ‘e’, ‘i’, ‘o’ and ‘u’. For example, if the user enters the word ‘hello’ the output from option 6 will be ‘hll’. If the user enters the word ‘Eamonn’ the output will be ‘mnn’. Add the method to the switch block so that when the user chooses option 6 from the menu, their word is output with all the vowels removed. [3 marks]

6. Currently the program quits after the user has entered their menu choice. Write a loop so that once the user has entered their choice, they are shown the menu again, and can choose another menu item. This loop should continue until the user chooses option 8 (Quit). Please note that the word that the user enters should be operated on by each option that they choose, not the word after it has been altered by one or other of the methods.

   Make sure that if the user entry is invalid; that is, a number that is not 1–8, or a `String`, they are allowed to enter another choice, and that this will continue until they manage to enter a valid menu choice. [3 marks]

7 Write another loop, so that when the user chooses option 8 (Quit), instead of the program ending they are asked if they wish to enter another word (yes/no). If they input ‘y’ or ‘Y’ (or anything that starts with ‘y’ or ‘Y’) then they are again asked to enter a word, and then shown the menu again and are back in the inner loop, and able to choose as many menu choices as they wish before choosing to quit; at which point they will be asked if they wish to enter another word. This should continue until the user enters a response that is not ‘y’ or ‘Y’, or does not start with ‘y’ or ‘Y’. [3 marks]

Reading for part a.

- http://docs.oracle.com/javase/7/docs/api/java/lang/String.html

Deliverables for part a.

- An electronic copy of your revised program: *StringInput.java.*

PART B

Consider the class, *ReceiptBefore.java*, file. When you run it (with the *receipt info.txt* file in the same directory) you should get a file as output, *formatted receipt.txt.*

In this part of the coursework assignment you are required to write the class *ReceiptAfter.java.* This class should do exactly what the *ReceiptBefore* class does, only it should do it using methods, and a constructor. Note that each of your methods should do only one thing and that constructors are used to initialise instance variables. Also note that instance methods can operate on instance variables, unlike static methods. That is to say, a method that is going to access or change an instance variable should be an instance method.

Static methods

Static methods can be used for methods that do not use any instance variables. They are most often used for utility methods; that is, methods that perform useful tasks that may have wider application than just inside the class. For example, the `Collections` class provides static methods that can be used by any class that implements the `Collection` interface (this includes `Vector` and `ArrayList`). These static methods do useful things such as copying, searching and sorting. See:
http://docs.oracle.com/javase/7/docs/api/java/util/Collections.html

Static variables

Static variables are variables that are the same for every instance of the class. For example, suppose your class has `static int eg = 5`, and a static method to update the `int`. Every object is going to see `eg = 5` and when you use the method to update the `int,` every object will see the new value.

List interface

You will note that in the file you have been given, ArrayLists are declared to be of type `List`, which is an interface: http://docs.oracle.com/javase/7/docs/api/java/util/List.html

Explanation below, modified from Effective Java, 2nd edition by Joshua Bloch

> You should use interfaces rather than classes as parameter types. More generally, you should favour the use of interfaces rather than classes to refer to objects. If appropriate interface types exist, then parameters, return values, variables, and fields should all be declared using interface types.
>
> Get in the habit of typing this:
>
> // Good - uses interface as type
> `List<Subscriber> subscribers = new ArrayList<Subscriber>();`
>
> rather than this:
>
> // Bad - uses class as type!

```
ArrayList<Subscriber> subscribers = new ArrayList<Subscriber>();
```

If you get into the habit of using interfaces as types, your program will be much more flexible. If you decide that you want to switch implementations, all you have to do is change the class name in the constructor.

For example, the first declaration could be changed to read:

```
List<Subscriber> subscribers = new LinkedList<Subscriber>();
```

and all of the surrounding code would continue to work. The surrounding code was unaware of the old implementation type, so it would be oblivious to the change.

1. Write the *ReceiptAfter* class. Your class should have the same output as the *ReceiptBefore* class. You should not need to change any of the statements in the class, but should start by considering how to break the class into methods, remembering that each method should only do one task. Once you have written your methods the main method should be empty.
   NOTE: You may choose to write your program only with static methods, in which case the maximum mark you can achieve for part (b) is 19. [15 marks]

2. Decide which of your methods should be static and which should be instance methods, and implement this. [4 marks]

3. Write a constructor for your class (if your class has instance methods). [4 marks]

4. Use the main method to test the output of the class. If you have written a constructor and instance methods you should make an object and use dot notation to test the class. [2 marks]

5. Write a comment at the start of the class briefly explaining what the program does. Write a short comment at the start of each method explaining what the method does. If there are parts of your program that do not work as you intended you should note this in your comments. [2 marks]

Reading for part b.
Volume 2 of the CO1109 subject guide; in particular:

- Chapter 7 (reading and writing to files)
- Chapter 9 (defining classes)
- Chapter 11 (exception handling)
- Chapter 12 (vectors – as arraylists and vectors both implement the `List` interface; the programs in the chapter should work if an `ArrayList` is substituted for a `Vector`).

The Java API, including:

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

Useful links on static methods:

- https://docs.oracle.com/javase/tutorial/java/javaOO/classvars.html
- http://javarevisited.blogspot.co.uk/2013/07/when-to-make-method-static-in-java.html

Deliverables for part b

- An electronic copy of your program: *ReceiptAfter.java.*
- An electronic copy of any other Java classes that you wrote in order to complete the assignment (optional)

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 buried inside a directory structure. This amounts to 50 marks altogether. There are another 50 marks available from Coursework assignment 1.

Total marks for part a.	[21 marks]
Total marks for part b.	[27 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]