Computing and Information Systems/Creative Computing
University of London International Programmes
CO2220 Graphical Object-Oriented and Internet programming in Java
Coursework assignment 2 2016–2017

Introduction

This is Coursework assignment 2 (of 2 coursework assignments in total) for 2016–2017. Part (A) asks that you demonstrate an understanding of parsing with *String.split(),* writing to and reading from a file and serialization. Part (B) looks at *String.format()* and the Comparator interface in the context of a client/server system.

IMPORTANT NOTE: Please use the most recent version of Java, Java 8.

Electronic files you should have:

Part A

- *TextQuestion.java*
- *TextQuestionUtils.java*
- *testquestions.txt*

Part B

- *Client.java*
- *ClientGUI.java*
- *CourseworkComparator.java*
- *History.java*
- *ServerUtils.java*
- *Student.java*
- *StudentFileReader.java*
- *StudentServer.java*
- *StudentServerEngine.java*
- *TotalComparator.java*
- *marks.txt*

What you should hand in: very important

At the end of each section there is a list of files to be handed in – **please note the handing-in requirements supersede the generic University of London instructions**. Please ensure that you hand in **electronic versions** of your .java files since you cannot gain any marks without handing them in. Class files are **not** needed, and any student handing 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**.

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 files that are **not** contained in a directory;

students who upload their files in a directory can only achieve 49/50 marks.

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

The examiners intend to compile and run your Java programs; for this reason, programs that do not compile will not receive any marks.

Your are asked to give your classes certain names; please follow these instructions carefully. Make sure there is no conflict between your file name and class names. Remember that if you give your files names that are different from the names of your classes (*e.g. cwk1-partA.java* file name versus *TextQuestionUtils* (class name) your program will not compile because of the conflict between the file name and the class name.

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 coursework assignment.

List interface

You will note that in the files you have been given, `ArrayLists` are declared to be of type `List`, as are methods that return an `ArrayList`. `List` 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.

CO2220 Coursework assignment 2
Part A

Consider the *TextQuestionUtils* class. The output of the completed class should be:

```
/******* parsed from text file testquestions.txt *******/
Is it true that event sources (such as a JButton) can be only
be registered with one event handler? no
What is the class that all exceptions sub-class?
java.lang.Exception
Should Swing applications directly call the paintComponent()
method? no
Java has two types of stream; chain is one, what is the other?
connection
Is a child class serializable if the parent class is? yes
How many areas does BorderLayoutManager have? 5

/******* parsed from text file written by
TextQuestionUtil.writeToFile() *******/
Is it true that event sources (such as a JButton) can be only
be registered with one event handler? no
What is the class that all exceptions sub-class?
java.lang.Exception
Should Swing applications directly call the paintComponent()
method? no
Java has two types of stream; chain is one, what is the other?
connection
Is a child class serializable if the parent class is? yes
How many areas does BorderLayoutManager have? 5

/******* deserialised from a file *******/
Is it true that event sources (such as a JButton) can only be
registered with one event handler? no
What is the class that all exceptions sub-class?
java.lang.Exception
Should Swing applications directly call the paintComponent()
method? no
Java has two types of stream; chain is one, what is the other?
connection
Is a child class serializable if the parent class is? yes
How many areas does BorderLayoutManager have? 5
```

You are tasked with completing the methods in the *TextQuestionUtils* class such that the output in your revised class is the same as that above. Please do not change the method headings, the *main* method or the *test()* method as you complete the following tasks:

1. You are given the heading of the *serializeToFile(String, List<TextQuestion>)* method. Complete the method. In your answer make sure that the method does what the comments written above the heading say it will do. [7 marks]

2. You are given the heading of the *deserializeFromFile(String)* method. Complete the method. In your answer make sure that the method does what the comments written above the heading say it will do. [7 marks]

3. The *readFromFile(String*) method uses a method called *parseQuestion(String)* to parse a `String` to a `TextQuestion` object using *String.split()*. Complete the *parseQuestion(String)* method. In your answer make sure that the method does what the comments written above the heading say it will do. [7 marks]

4. You are given the heading of the *writeToFile(String, List<TextQuestion>)* method. Complete the method. In your answer make sure that the method does what the comments written above the heading say it will do. [7 marks]

Reading for Part A
Note that in the list below, the 'Subject guide' refers to volume 2, and HFJ refers to Head First Java

- Subject guide, Chapter 2, Sections 2.2, 2.11 and HFJ 275–78 (static utility classes)
- Subject guide, Chapter 3, Sections 3.2, 3.3, 3.5, 3.6 and HFJ 319–26 and 329–33 (handling exceptions)
- Subject guide, Chapter 5, Sections 5.3 and HFJ 447, 452–54 and 458–59 (files, parsing with *String.split(),* streams including *BufferedReader* and *BufferedWriter*)
- Subject guide Chapter 6, sections 6.2, 6.3, 6.4 and HFJ 431–38 and 441–43 (serialization)

Deliverable for Part A

- An electronic copy of your revised class: *TextQuestionUtils.java*

Please put your name and student number as a comment at the top of your Java file.

Part B

Testing the *ClientGUI*

You should compile and run the *ClientGUI.java* file. To do this you will need to first compile and start the *StudentServer.java*. You can run both programs from the shell (cmd.exe), but you will need to open the shell twice, once to get the *StudentServer* started and another to run the *ClientGUI*. Note that the *ClientGUI* has been hard-coded to connect to a local host.

When you start the *ClientGUI* class you should see a simple GUI with a *JButton,* a scrollable *JTextArea* and a *JTextField*. When you click on the 'Connect' button, if you see the message 'ERROR: Connection refused: connect', and you have the *StudentServer* class running, it may be that you need to tell your firewall to allow java through it. If this is the case you will have to restart the *StudentServer* and the *ClientGUI* once you have adjusted your firewall. Whenever you disconnect from the *ClientGUI* you will have to restart the *StudentServer* from the shell (cmd.exe) as the server is single-threaded and dies once the client has disconnected.

What you should see when you run the *ClientGUI* is the following, on the *JTextArea*:

```
Welcome to the Student Results Server. Available commands:
SHOW ALL          show all student results
SORT EXAM         sort student results by exam mark
SORT CWK          sort student results by coursework mark
SORT GRADE        sort student results by grade
SORT TOTAL        sort student results by total (final) mark
GRADE <grade>     show all students that achieved a grade of <grade>
SEARCH <name>     show all students that have <term> in their name
SHOW HELP         show this help
```

Enter each of the listed commands to test the *ClientGUI*. You will note that the *Student* class has five instance variables: *name, exam, cwk, total* and *grade.* 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 *Student* class implements the rule that a student passes a module if they have achieved a mark of at least 35 in the coursework and exam components, with a weighted average mark of both components (the *total* field) of 40 or more. *Student* objects therefore have five fields that can be searched for in a list of *Student* objects, and that a list of *Student* objects can be sorted by.

The ClientGUI has commands to sort the *List* of *Student* objects by EXAM (the `int` *exam* field), by CWK (the `int` *cwk* field), by GRADE (the `String` *grade* field) and by TOTAL (the `int` *total* field).

You will notice that two of the SORT commands do not work. Consider the *StudentServerEngine* class. It has a *sort(String)* method (see below), where two statements have been commented out because the methods that they call in turn rely on *Comparator* classes that have not been written yet.

```
private String sort(String sortBy) {
   switch (sortBy.toLowerCase()) {
         case "cwk":         return sortByCoursework();
         case "total":       return sortByTotal();
         //case "exam":      return sortByExam();
         //case "grade":     return sortByGrade();

         default:  return "I don't know how to sort by that!";
   }
}
```

Please complete the following tasks:

1. The *Student* class has a basic *toString()* method. Write an improved *toString()* using *String.format()* to set out the fields of the class in a readable way. [5 marks]

2. Write the *ExamComparator* class. Once you have written the class, remove the comment marks ("//") from the appropriate statement in the *sort(String)* method of the *StudentServerEngine* class and run the *ClientGUI* to test your new *Comparator*. [7 marks]

3. Write the *GradeComparator* class. Once you have written the class, remove the comment marks ("//") from the appropriate statement in the *sort(String)* method of the *StudentServerEngine* class and run the *ClientGUI* to test your new *Comparator*. [8 marks]

Reading for Part B

- Head First Java pp.294–97 for more about *String.format()*
- http://docs.oracle.com/javase/7/docs/api/java/lang/String.html (scroll down or search to find a description of the *String.format()* method).
- Chapter 7 of Volume 2 of the CO2220 subject guide, pp.63–66 (clients and servers).
- *Head First Java* pp. 473–85 (clients and servers)
- You are expected to do some research to help you complete Questions 2–4 but see:
  http://stackoverflow.com/questions/7117044/how-to-use-java-comparator-properly
  https://docs.oracle.com/javase/7/docs/api/java/util/Comparator.html

Deliverables for Part B

Please submit an electronic copy of the following:

- *Student.java* (with revised *toString()* method)
- *ExamComparator.java*
- *GradeComparator.java*

Please put your name and student number as a comment at the top of your Java files.

MARKS FOR CO2220 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	[28 marks]
Total marks for Part B	[20 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]