Computing and Information Systems/Creative Computing
University of London International Programmes
CO2220 Graphical object-oriented and internet programming in Java
Coursework assignment 2
2015–16

Introduction
This is Coursework assignment 2 (of two coursework assignments total) for 2015–16. In part (a) you are asked to demonstrate an understanding of static utility classes, inheritance, constructors, exceptions, files, serialization, arraylists and their methods, String.format(), the StringBuilder class, toString() methods and testing. Part (b) covers serialization (again), network programming (clients and servers) and more advanced GUIs than the simple one seen in Coursework assignment 1.

Electronic files you should have:

Part a.
- *Grocery.java*
- *fruits.csv*

Part b.
- *Book.java*
- *BookClient.java*
- *BookFileParser.java*
- *BookServer.java*
- *BookServerEngine.java*
- *ServerUtils.java*
- *ClientGUI.java*
- *History.java*
- *books.ser*

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 – that is, students who upload their directories can only achieve 49/50 marks.

There is one mark available in part (a) for giving your methods and classes the names that you have been asked to give them.

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 for**

incorrect submission. Please make sure all Java files compile, even if they do not work entirely as they should. **Candidates will not receive any marks for Java programs that do not compile.**

CO2220 Coursework assignment 2

PART A

Consider the *Grocery* class.

1. Write a *toString()* method for *Grocery*. The method should use *String.format()* to output the fields of the object in a readable way. You should take care to ensure that if more than one object is being printed (for example, from an ArrayList) then the fields of each object should be clearly visually distinguishable from the fields of the next. https://docs.oracle.com/javase/7/docs/api/java/lang/String.html [2 marks]

2. Write a *Fruit* class, as a sub-class of *Grocery*. It should have only one field, *pricePerKilo*, a `double`. [2 mark]

3. Write the following constructors for your *Fruit* class: [4 marks]

   - a constructor that takes three parameters. Use the *super* keyword (2 marks).
   - a constructor that only takes two parameters; that is, it does not have a *pricePerKilo* parameter. Use the keyword *this*, and set the value of *pricePerKilo* to –1 (2 marks).

4. Write a *toString()* method for *Fruit*. The method should (1) use *String.format()* and (2) call on the *toString()* method in the superclass to do some of the work. [3 marks]

5. Write a static utility class, called *FruitUtils*. Your class should have the following five static methods: [14 marks]

   - *displayFruits()*: A method to display an arraylist of *Fruit* objects to the screen (2 marks). **The method should use your *toString()* method for the *Fruit* class.**
   - *parseFruitsFromCSV()*: A method to read in from a CSV file, and make *Fruit* objects from the lines in the file. The method should store the *Fruit* objects in an arraylist. Make sure to test this method with the *fruits.csv* file given to you. Do not edit the *fruits.csv* file. (3 marks).
   - *writeFruitsToFile()*: A method to output an arraylist of *Fruit* objects to a text file, formatted to be readable; namely, each *Fruit* object's fields should be labelled, easy to read and clearly distinguishable from the next *Fruit* object's fields (2 marks). Use

UTF-8 encoding (1 mark).

- *serializeToDisk()*: A method to serialize an arraylist of *Fruit* objects to a file (3 marks).
- *fromSerialized(): A method to deserialize from the file created by your serializeToDisk() method. The method should produce an arraylist of Fruit objects*

6. Make sure that all exceptions likely to be thrown by your methods, are handled within your methods with try/catch. [4 marks]

7. Write a constructor for your *FruitUtils* class. Choose an appropriate access level. [1 mark]

8. Write a *FruitTest* class. In the main method of the class write statements to test the classes and methods that you have written. [4 marks]

9. Make sure to give your classes and methods the names that you have been asked to give them. [1 mark]

Reading for part a.
Note that in the list below, subject guide refers to CO2220 Volume 2, and HFJ refers to the *Head First Java* textbook.

- Subject guide Chapter 2, sections 2.2, 2.11 and HFJ 275–278 (static utility classes).
- Subject guide Chapter 3, sections 3.2, 3.3, 3.5, 3.6 and HFJ 319–326, and 329–333 (handling exceptions).
- Subject guide Chapter 5, sections 5.3 and HFJ 447, 452–454 and 458–9 (files).
- Subject guide Chapter 6, sections 6.2, 6.3, 6.4 and HFJ 431–8; 441–3 (serialization).
- HFJ 294–301 (String.format()).
- HFJ 247–256 (constructors).
- http://www.wideskills.com/java-tutorial/java-tostring-method (*toString()* methods)

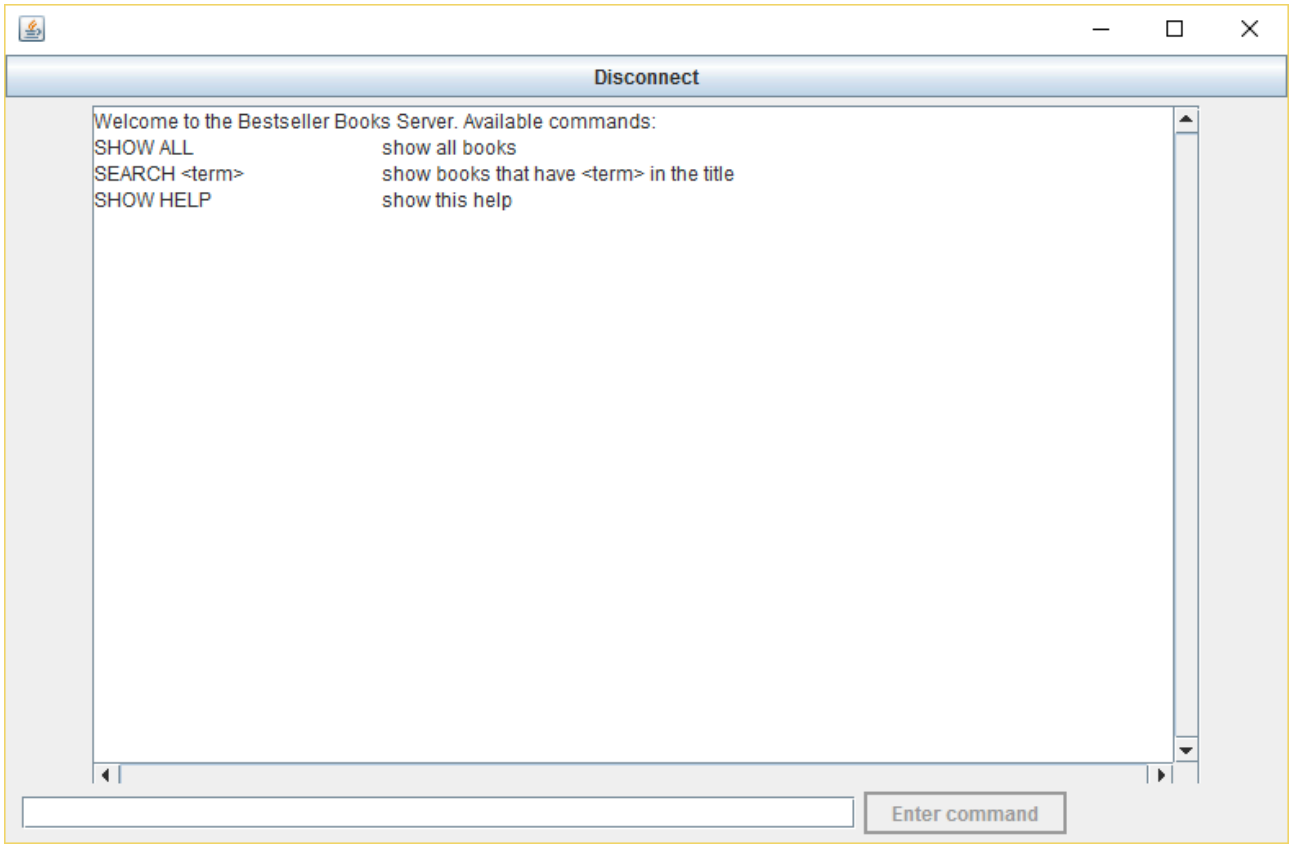
Deliverables for part a.

- *Grocery.java* (with added *toString()* method)
- *Fruit.java*
- *FruitUtils.java*
- *FruitTest.java*

Please make sure that all Java files compile – no marks can be given for files that have compilation errors.

PART B

Testing the ClientGUI
You should compile and run the *ClientGUI.java* file, and after connection you should see the simple GUI as in the image below, which has two *JButtons,* a scrollable *JTextArea* and a *JTextField*.



To do anything useful with *ClientGUI* you must first compile and run *BookServer.java* so that you have a server for your client to connect to. You can run both programs from the shell (cmd.exe) but will need to open the shell twice, once to get the *BookServer* started and once to run the *ClientGUI*. Note that the *ClientGUI* has been hard-coded to connect to a local host. When you click on the 'Connect' button, if you see the message 'ERROR: Connection refused: connect', and you have the *BookServer* 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 *BookServer* and the *ClientGUI* once you have adjusted your firewall. Whenever you disconnect from the *ClientGUI* you will have to restart the *BookServer* from the shell (cmd.exe) as the server is single-threaded and dies once the client has disconnected.

The search() method
You should run the *ClientGUI* class, enter the commands that the GUI will accept and see what happens. You should notice that the SEARCH <term> command does not work. It is intended that the command will work as follows.

When the user enters a search `String`, the search function will look for a book, or books, with that search `String` in its title. This includes checking to see if the `String` given forms any part of any of the words in a book's title. All books found, or an error message as appropriate, are returned to the *JTextArea*. For example:

`search martian` will return:
The Martian by Andy Weir. 384 pages. Genre: Sci-Fi. Published 2015. ISBN: 1785031139. Sales: 101823.

`search rl` will return:
Gone Girl by Gillian Flynn. 512 pages. Genre: Literature. Published 2014. ISBN: 1780228228. Sales: 97231.
Guinness World Records 2016 by Guinness World Records. 256 pages. Genre: Non Fiction. Published 2015. ISBN: 1910561010. Sales: 79918.

`search zebra` will return:
no results found

Please note that the *search()* method should perform a case-insensitive search.

The History and HistoryListener classes
You will note that the *ClientGUI* has a private *HistoryListener* class. You have also been given an empty *History* class. You will also notice that line 180 has been commented out in the *ClientGUI*: `//history.add(field.getText());`

When the *History* class and the *HistoryListener* have been completed, and the comment marks removed from line 180, the *ClientGUI* is supposed to work as follows:

Any command entered by the user is stored as a `String`. When the user presses the up arrow, the most recently entered command is shown in the *JTextField*. Further presses of the up arrow will display other commands in the *JTextField*, ordered by the most recently entered until there are no more commands to display or the user stops. At any time the user can press the down arrow, in which case the commands already displayed will be gone back through, ending with an empty *JTextField*. In this way the user can find any command they have already entered and choose to enter it again, or to edit it before entering it.

Note that pressing the down arrow before first pressing the up arrow will have no effect.

For example, suppose that the user has typed in three commands: "show all"; "show help"; and "search me".

The user clicks "enter command" after typing "search me" and so has an empty *JTextField*. Pressing up once will bring back "search me". Pressing the up arrow again will recall "show help" to the *JTextField*. If the user now presses the down arrow the *JTextField* will display "search me" and pressing down again brings the user back to an empty *JTextField*.

1. Write the *search()* method, add it to the *BookServerEngine* class, and then make any other necessary changes to *BookServerEngine*, so that the user can now search book titles as described above. [6 marks]

2. Remove the comment marks from line 180 in the *ClientGUI* class, and complete the *History* and *HistoryListener* classes, so that the *ClientGUI* works as described above. [7 marks]

Reading for part b.

- Chapter 7 of Volume 2 of the CO2220 subject guide, pages 63–66 (clients and servers).
- *Head First Java* pages 473–485 (clients and servers).
- *You are expected to do some research to help you complete Question 2.*

Deliverables for part b.

- *BookServerEngine.java (with added search() method and any other necessary changes).*
- *History.java (completed).*
- *ClientGUI.java (with completed HistoryListener class).*

Please make sure that all Java files compile – no marks can be given for files that have compilation errors.

MARKS FOR CO2220 COURSEWORK ASSIGNMENT 2

The marks for each section of the Coursework assignment 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 allows 50 marks altogether. There are another 50 marks available from Coursework assignment 1.

Total marks for part a.	[35 marks]
Total marks for part b.	[13 marks]
Mark for giving in uncompressed Java files	[1 mark]
Mark for giving in standalone Java files; namel, files **not** enclosed in a directory	[1 mark]
Total marks for Coursework assignment 2	**[50 marks]**

[END OF COURSEWORK ASSIGNMENT 2]