ISY00246 – Client/Server Systems

Assignment 1 (S2 2015)

This assignment is due Friday in week 8 (14th August). Your tutor may provide additional submission information to you. You should submit your source code (Java and HTML) for the all parts of the assignment via e-mail to your tutor (or by other means approved by your tutor). Note that marks will be deducted for poorly structured or uncommented code. All source code files submitted must include title comments that at least identify the author and the assignment part. The separate parts of the assignment are to be submitted in separate subdirectories (e.g. Part1, Part2 etc.) – submissions that ignore this instruction and leave all assignment files in one directory will be penalised.

Please note that this assignment will be due 2-3 weeks after the suggested study timetable has covered the last of required materials. Do not leave this assignment to the last minute – do it while concepts are fresh in your mind. If you require an extension you must apply to your tutor before the due time to be considered.

Part 1 – Java Programming (9 marks – Topics 1-2)

This part of the assignment is for you to demonstrate your understanding of Java programming.

a. (**3 marks**) Write a program to read integer numbers from the keyboard two at a time at a time until either ten numbers are input or a blank line is input. The program should read the numbers into an `int` array. After the numbers are read the program should print the number of numbers (e.g. "`There were 6 numbers entered`"), then print the string "`Maximum =`" followed by the maximum of the numbers entered, and finally print the string "`Minimum =`" followed by the minimum of the numbers entered. An example output is:

```
C:\> java Numbers
1 2
-2 3
3 4
4 5

There were 8 numbers entered.
Maximum = 5
Minimum = -2
```

One way to implement this program is to use the `substring()`, `indexOf()` and other member functions of the `String` class. You should also check for illegal input, i.e. non-numbers or junk following a number. If a non-number is entered then your program should display an error message and re-prompt for another number. If the maximum number of numbers is entered (i.e 10) then print a message stating that this happened before continuing with the summary messages.

b. (**2 marks**) Consider the following interface designed for a shop's stock objects:

```
public interface Stock {
      public int getID();
          // get stock ID
      public String getDescription();
          // get stock description
      public int number()
          // get number of items
      public void setNumber(int num)
          // update number in stock
      }
```

Write a Java class called `Confectionary` that implements this interface. Instances of this class should be able to modify the other number of stock but **not** the ID or description. The `Confectionary` class should *also* hold a string representing product expiry date. The product ID and description should be immutable, i.e. the ID and description are established

when objects of this class are created via a constructor implemented in the `Confectionary` class and never changed after that.

Test this class by implementing a test program that creates objects of this class by creating at least three objects. This program will be used when marking your assignment.

c. **(1 mark)** Implement another class `SoftDrink` that implements the `Stock` interface. This class also stores a `package` number that indicate the number of items in the package. It does not have expiry data. Again, make the ID and description immutable. Also make the package data immutable with a single method for retrieving the package number.

Now create a test program with a **single** polymorphic method that takes any object with the `Stock` interface (e.g. `Confectionary` object or `SoftDrink` object) as a parameter and returns a string with stock ID and description. Test it by passing two objects of different classes.

d. **(3 marks)** Now implement a program that takes a file of `Confectionary` and `SoftDrink` data in **randomly** mixed records in a file. The file name will be given on the command line as `args[0]`. You will need to work out how to identify the different types of data (i.e. separate confectionary from soft drink data) in the file, e.g. by inserting a flag at the start of each record. You can assume a maximum number of records if you wish. The program stores the data in a data structure in appropriate objects (an array of objects will do but there are better container classes available that make programming easier).

After reading in all the data, print out all of the `Confectionary` items and then all of the `SoftDrink` items. Print all of the attributes for each object types one per line.

An example output may be as follows. You may like to improve this format somewhat.

```
Confectionary:
123, Red lollies, Feb 2010
127, Mars Bars, 1/12/2020
130, Big chockies, 2012

Soft Drinks:
128, Black anon, 24
1002, Red hyper, 6
1004, Moo juice, 1
1011, Green Soother, 24
```

Hint: Create the file using the text editor and use the `readLine()` member of a `BufferedReader` object (or similar) to read the data files into your data structure.

Part 2 CGI systems (3 marks – Topic 3)

The aim of this part of the assignment is for you to demonstrate your understanding of CGI programs by developing an HTML form. Although you will not be required to write CGI server programs, you are asked here to interpret a given CGI program. Use of fancy HTML is appreciated but will not earn extra marks. However, correct and consistent use of the tags used in the tutorials is expected, as is sufficient prompts and headings for the user to interpret the form.

A Java CGI program called `Assignment1_2` is installed at:

`http://spike.scu.edu.au/cgi-bin/Assignment1_2/Assignment1_2`

This source code is available on the MySCU site. Your task is to implement a HTML form to interface with the program. This means that you will be tested on your understanding of Java as well as your understanding of HTML forms and CGI.

In this part you will need to exercise all of the program inputs using a form with several buttons and simple text fields. Note that the CGI program will build the output to be displayed in HTML.

Part 3 Internet Sockets (8 marks – Topic 4 & 5)

The aim of this exercise is for you to implement both datagram and stream sockets. There are separate parts for each kind of socket. You should refer to the laboratory exercises for this question as answers here extend some of the examples in the notes.

a. (**Datagram Sockets – 3 marks**) write a client program that uses port 2014 for connecting to the Internet and server program that will listen to port 2015. We will be sending messages from the client program to the server program as datagrams containing the string "yes" or "no". The client program should be run with the server address as a command line argument and prompt for a "yes" or "no" vote as in the following example:

```
> java Vote 203.2.60.27

Enter "yes" or "no": yes
Vote "yes" sent to 203.2.60.27
```

Note that for Netbeans users you simply need to add the command line arguments (Run/Set project configuration/Customize/Run/Arguments) and use the normal compile and run features as demonstrated in the Collaborate sessions.

The client should then exit. You will need to insert the correct IP address of the server program that you have already started. Your program should allow any combination of upper or lower case to be entered. If an error occurs, e.g. an exception thrown, then the status message (i.e. where "Vote X sent …" is above) should instead display the problem. Also, if the user types in an incorrect string, an error message should be displayed and the user should be prompted again for input.

The server program should be run as follows with hard-coded port number 2015. The program will display the vote it receives and keep a running tally of the votes from all clients as in the example run below:

```
> java VoteServer
Voting server 203.2.60.27 active on 2015.
Type Ctrl-C to finish.

[203.2.60.17] "Yes" vote received
   So far Yes= 1  No= 0.
[203.2.60.22] **Error ** Bad vote string received
[203.2.60.19] "No" vote received
   So far Yes= 1  No= 1.
…
```

Here three messages have been received, from hosts 203.2.60.17, 203.2.60.22 and 203.2.60.19. An incorrect string is received from one of the hosts (implying a bug in the client!). The vote tally is displayed after each valid vote is received. Notice that the server prints its own server address and port number when it starts so you can tell someone else your Internet address or discover your own.

Note also that your client and server programs must be capable of displaying the output **exactly as shown above**. Also, you will need to handle incorrect arguments on the client's command line, which help you to learn about command lines.

b. (**Stream Sockets – 3 marks**) Write client and server programs (`NameClient` and `NameServer`) that operate on a port you select and act as client and server programs for accessing names on another computer. The client program should accept the server IP address as a command line argument and be run as in following example:

```
> java NameClient 127.0.0.1
```

Once started, the client program should then display a menu as follows:

```
Name Saver Server Menu

1. Add a name
2. Remove a name
3. List all names
4. Check if a name recorded
5. Exit

Enter selection [1-5]:
```

The menu should be displayed continually until option 5 is selected. The client program should prompt the user for the name in options 1, 2 and 4. A name will be any string that can be typed in one line. Spaces in the name are allowed.

The server program should accept commands on its input stream and send replies on its output stream. The client will display the server's response. In the case of option 3, the client will also print a status message regarding the number of names displayed after displaying the separate names, e.g.:

```
Total names displayed = 21.
```

The name server should send an error status back to the client if a problem occurs, e.g. when a name does not exist for options 2 and 4.

You will need to implement a data structure on the server side to save all the names. A good implementation will not limit the number of names and make the server persistent, i.e. when the server is restarted the server would remember the names from previous sessions. However, do this last and just get the server to work before worrying about saving names to a file for persistence. (Hint: the easiest way to store the names in a file is one per line)

Note that menu selection 4 is the most difficult so leave it until last. You have to figure out how the server will notify the client that it has sent all of the name data to display and how to indicate error messages.

Marks will be allocated for just the menu working and for individual menu choices. Even though menu selection 4 is the most difficult to implement it is not worth any more marks than the other menu selections.

c. (**Threads – 2 marks**) Rewrite part b. to accept multiple connections at the same time. You will use threads to implement a multithreaded server (just like the real thing). It would be best to test this with clients (from part b.) running on several machines. However, if this is not possible, you can still test it effectively using several copies of the client program on the one machine by having several menus active in different windows. You will need to consider race conditions with your name data store and insert locking mechanisms **if** necessary (it will not be necessary for some designs).