CS421 - Computer Networks
Programming Assignment 2
VendingMachine
Spring 2016

Due: 24/04/2016

In the first programming assignment, you were asked to implement a server that simulates a vending machine and a client that requests service from the server in Java. Now you will extend your program to allow for multiple clients to connect to the server, simultaneously. Again your program must be a console application (no graphical user interface (GUI) is required) and should be named as *VendingMachine* (i.e., the name of the class that includes the main method should be *VendingMachine*). Your program should run with the

```
java VendingMachine [<IP_address>] <port_number>
```

command, where `<IP_address>` is the IP address of the host to which your program should connect and `<port_number>` is the port number to which the socket is bound. The first command line argument, i.e., `<IP_address>`, is **optional**. If it is not given, the executed program behaves as the server. It opens a server socket at port `<port_number>` and listens for a connection. If it is given, the executed program behaves as the client. It connects to the host with IP address `<IP_address>` and port `<port_number>`.

The client The client simulates a customer getting service from a vending machine. There are two types of messages that the client may send to the server.

- **GET ITEM LIST**
  The client sends

  ```
  GET ITEM LIST\r\n
  \r\n
  ```

message to the server to request the details of the items in the vending machine.

- **GET ITEM**
  The client sends

  ```
  GET ITEM\r\n
  <item_id> <item_count>\r\n
  \r\n
  ```

  message to the server to get `<item_count>` number of items with id `<item_id>`. The fields `<item_id>` and `<item_count>` should be separated by one space character.

The server The server simulates a vending machine. When the server starts, it reads the details of the items from the file `item_list.txt` residing in the folder including the source file(s). Each line of this file includes the **unique** integer id and the name of an item, and the number of items with that id in the virtual vending machine. After the file is read and the details of the items in the machine are obtained, the server listens to the server socket opened at port `<port_number>`. When a client is connected to the server, the server waits for the client to send a message and respond the client with an appropriate message and also starts a new thread to listen for another client. There are three types of messages the server may send to the client.

- **ITEM LIST**
  If the server receives a `GET ITEM LIST` message, it responds to the client with an `ITEM LIST` message including the details of the items in the machine. Assuming that there are M types of items in the machine and the machine includes `<item_count_m>` items with item id `<item_id_m>` and item name `<item_name_m>` for $m = 1, \ldots, M$, then the following message is sent to the client:

  ```
  ITEM LIST\r\n
  <item_id_1> <item_name_1> <item_count_1>\r\n
  ...
  <item_id_M> <item_name_M> <item_count_M>\r\n
  \r\n
  ```

The fields `<item_id_m>`, `<item_name_m>`, and `<item_count_m>` should be separated by one space character for $m = 1, \ldots, M$.

- **SUCCESS**
  If the server receives a `GET ITEM` message and the machine includes the requested items, the following message is sent to the client:

  ```
  SUCCESS\r\n
  \r\n
  ```

- **OUT OF STOCK**
  If the server receives a `GET ITEM` message and the machine does not include the requested number of items of that type, the following message is sent to the client:

  ```
  OUT OF STOCK\r\n
  \r\n
  ```

Note that since several clients may request the same item, the requests should be processed in order.

Other issues

- The file including the initial item list should reside in the folder including the source file(s) and be named as `item_list.txt`. This file is read once when the server starts. The fields of the file needs to be separated by space or tab characters. You may assume that each line of the file includes a different type of item and the number of items is positive.
- The name of an item should not include a space character.
- The message to be sent by the client is determined by the user input interactively, that is, the client should prompt the user to get input. The format of user input is not specified; so, you may choose an appropriate format.
- The server should send an `OUT OF STOCK` message if the client requests an item with id that is not found in the `item_list.txt` file.

- The server should keep track of items remaining. After each purchase the server should update the `<item_count>`, respectively and in the next `item_list.txt` command the updated version of the items will be displayed. Optionally, when the server runs out of all items, the server can halt.

- The server should print received messages and sent messages to the command line.

- The client should prompt the user to terminate the connection. When the user wants to terminate the connection, the client should summarize the details of the items successfully received from the machine and terminate the connection.

- The connection between the server and a client is terminated when the client program ends. The server should handle an exception thrown when the client program terminates.

- When the connection between a client and a server is initiated, the server should listen for another client connection in a new thread. Optionally you can hard-code a limit to the maximum number of threads a server can handle.

- In order to access the list of items across all threads you should synchronize the list variables.

- Each thread runs as long as the client program runs. After the client program terminates, the server prints the overall result and the thread ends.

- Since the server needs to be implemented in a multi-threaded fashion and there is a single command-line for the server, the server should also indicate the ID of the client/thread for each line printed to the command line.

- The messages sent by the client and the server should be in the aforementioned format. You will lose points if the messages are sent in another format.

Example

Assume that the content of the `item_list.txt` be as follows

```
101   cola          15
102   diet-cola      8
103   chocolate     12
104   potato-chip    4
```

and let

```
java VendingMachine 9051
```

and

```
java VendingMachine 111.111.111.111 9051
```

be two respectively executed commands, where the former runs the server program with the first thread listening to port 9051 on a computer with IP address 111.111.111.111 and the latter runs a client program trying to connect to the server.

The first client requests the list of the items in the machine, respectively gets 3 cola items and 20 chocolate items. The server processes its request on the first thread and opens another thread upon new incoming clients. Then the following messages are sent:

1. client 1 → server

```
GET ITEM LIST\r\n
\r\n
```

2. server → client 1

```
ITEM LIST\r\n
101 cola 15\r\n
102 diet-cola 8\r\n
103 chocolate 12\r\n
104 potato-chip 4\r\n
\r\n
```

7. client 1 → server

```
GET ITEM\r\n
101 3\r\n
\r\n
```

8. server → client 1

```
SUCCESS\r\n
\r\n
```

9. client 1 → server

```
GET ITEM\r\n
103 20\r\n
\r\n
```

10. server → client 1

```
OUT OF STOCK\r\n
\r\n
```

Here a new client connects to the server on thread 2, requests to list the items and purchases 6 cola items. The flow of packets sent will be as follows.

1. client 2 → server

```
GET ITEM LIST\r\n
\r\n
```

2. server → client 2

```
ITEM LIST\r\n
101 cola 12\r\n
102 diet-cola 8\r\n
103 chocolate 12\r\n
104 potato-chip 4\r\n
\r\n
```

11. client 2 → server

```
GET ITEM\r\n
101 6\r\n
\r\n
```

12. server → client 2

```
SUCCESS\r\n
\r\n
```

After this, the first client requests to view the item list again. Note that the number of items in stock will be updated. The following messages are sent:

1. client 1 → server

```
GET ITEM LIST\r\n
\r\n
```

2. server → client 1

```
ITEM LIST\r\n
101 cola 6\r\n
102 diet-cola 8\r\n
103 chocolate 12\r\n
104 potato-chip 4\r\n
\r\n
```

The first client terminates the connection. And after some time the second client terminates too. Note that this sequence of events could happen in any order.

After the connection is terminated, the command-lines of the client will be the similar to the first assignment. However the server will output the log for each thread. Bellow you can see the console output of client 1 upon terminating the connection to the server.

Command-line of the client 1:
The connection is established.
Choose a message type (GET ITEM (L)IST, (G)ET ITEM, (Q)UIT): L
The received message:
  ITEM LIST
  101 cola 15
  102 diet-cola 8
  103 chocolate 12
  104 potato-chip 4
Choose a message type (GET ITEM (L)IST, (G)ET ITEM, (Q)UIT): G
Give the item id: 101
Give the number of items: 3
The received message:
  SUCCESS
Choose a message type (GET ITEM (L)IST, (G)ET ITEM, (Q)UIT): G
Give the item id: 103
Give the number of items: 20
The received message:
  OUT OF STOCK
Choose a message type (GET ITEM (L)IST, (G)ET ITEM, (Q)UIT): L
The received message:
  ITEM LIST
  101 cola 6
  102 diet-cola 8
  103 chocolate 12
  104 potato-chip 4
Choose a message type (GET ITEM (L)IST, (G)ET ITEM, (Q)UIT): Q
The summary of received items:
  101 3

And on client 2 we would have:

Command-line of the client 1:
The connection is established.
Choose a message type (GET ITEM (L)IST, (G)ET ITEM, (Q)UIT): L
The received message:
  ITEM LIST
  101 cola 12
  102 diet-cola 8
  103 chocolate 12
  104 potato-chip 4
Choose a message type (GET ITEM (L)IST, (G)ET ITEM, (Q)UIT): G
Give the item id: 101
Give the number of items: 6
The received message:
  SUCCESS
Choose a message type (GET ITEM (L)IST, (G)ET ITEM, (Q)UIT): Q
The summary of received items:
  101 6

The server will output the following logs to the console. Note that the messages from each client should not necessarily be sent consequently and might be received and processed in any order.

```
Command-line of the server:
item_list.txt is read
The current list of items:
   101 cola 15
   102 diet-cola 8
   103 chocolate 12
   104 potato-chip 4

Listening for client on THREAD1
**************************************
A client is connected on THREAD1
Listening for client on THREAD2
**************************************
Message received on THREAD1:
   GET ITEM LIST
Send the message:
   ITEM LIST
   101 cola 15
   102 diet-cola 8
   103 chocolate 12
   104 potato-chip 4
**************************************
Message received on THREAD1:
   GET ITEM
   101 3
Send the message:
   SUCCESS
**************************************
Message received on THREAD1:
   GET ITEM
   103 20
Send the message:
   OUT OF STOCK
**************************************
A client is connected on THREAD2
Listening for client on THREAD3
**************************************
Message received on THREAD2:
   GET ITEM LIST
Send the message:
   ITEM LIST
   101 cola 12
   102 diet-cola 8
   103 chocolate 12
```

```
   104 potato-chip 4
************************************
Message received on THREAD2:
   GET ITEM
   101 6
Send the message:
   SUCCESS
************************************
Message received on THREAD1:
   GET ITEM LIST
Send the message:
   ITEM LIST
   101 cola 6
   102 diet-cola 8
   103 chocolate 12
   104 potato-chip 4
************************************
THREAD1: The client has terminated the
connection.
The current list of items:
   101 cola 6
   102 diet-cola 8
   103 chocolate 12
   104 potato-chip 4
************************************
THREAD2: The client has terminated the
connection.
The current list of items:
   101 cola 6
   102 diet-cola 8
   103 chocolate 12
   104 potato-chip 4
************************************
```

Submission rules

You need to apply all of the following rules in your submission. **You will lose points if you do not obey the submission rules below or your program does not run as described in the assignment above**.

- The assignment should be submitted as an e-mail attachment sent to: **marzieh.eslami[at]cs.bilkent.edu.tr**.

  Any other methods (Disk/CD/DVD) of submission will not be accepted.

- The subject of the e-mail should start with *[CS421_PA1]*, and include your name and student ID. For example, the subject line must be

  [CS421_PA2]AliVelioglu20111222

  if your name and ID are *Ali Velioglu* and *20111222*. If you are submitting an assignment done by two students, the subject line should include the names and IDs of both group members. The subject of the e-mail should be

  [CS421_PA2]AliVelioglu20111222AyseFatmaoglu20255666

  if group members are *Ali Velioglu* and *Ayse Fatmaoglu* with IDs *20111222* and *20255666*, respectively.

- All the files must be submitted in a zip file whose name is the same as the subject line **except** the [CS421_PA2] part. The file must be a **.zip** file, not a **.rar** file or any other compressed file.

- All of the files must be in the root of the zip file; directory structures are **not** allowed. Please note that this also disallows organizing your code into Java packages. The archive should **not** contain:

  - Any class files or other executables,
  - Any third party library archives (i.e. jar files),
  - Any text files (including the `item_list.txt` file you have used in testing your program),
  - Project files used by IDEs (e.g., JCreator, JBuilder, SunOne, Eclipse, Idea or NetBeans etc.). You may, and are encouraged to, use these programs while developing, but the end result must be a clean, IDE-independent program.

- The standard rules for plagiarism and academic honesty apply, if in doubt refer to 'Student Disciplinary Rules and Regulation', items 7.j, 8.l and 8.m.