BTP400 Lab #4
Due: 11:58 pm, March 13 (Friday)

Part A
Focus: TCP/IP Socket-Based Network Programming.

You are asked to build a TCP/IP socket-based Java application that allows a Java client to send deposit/withdraw requests to a server. The server will then update an Account object sent by the client. (You may reuse and enhance the Account class that you have developed in the previous labs.)

Problem Description
Here are the interactions that will happen between the client and the server. When the client program is started, it displays on a console window all the details of a particular Account object. The client then allows a user to send requests to the server repeatedly. There are three types of requests, namely deposit, withdraw and quit. Here are some examples: “deposit 10.55”, “withdraw 100.15”, and “quit”:

After the client has sent a deposit/withdraw request to the server, it sends the Account object immediately to the server. Then it will wait for the server to send back the Account object whose balance has been updated. If the client has sent a “quit” request to the server, it will close the connection and terminate.

When the server program is started, it displays this message: “Welcome to Bank@Seneca!”. Then it receives and handles requests repeatedly. After the server has received a deposit/withdraw request, it will receive an Account object from the client. Then it will update the balance of the Account object according to the request. Finally it will send the Account object back to the client.

If the server has received a “done” request, it will close the connection and terminate.

Testing Requirements
1. You will run your client/server application on localhost.
2. You should use your full name as the name of the account. The starting balance of the account should be 1000.00.
3. You will use two command prompt windows to test your application.

Submission Requirements at Blackboard
Submit a) a jar file of your Java source code and b) one or more screenshots of the output.

Part B

Focus: Coordination of Multiple Threads (Without Data Sharing).

Problem Description

Create a multi-threaded application that uses threads to search the following two-dimensional array of strings:

```
String [][] data = {

        { "java", "I love java", "c++", "python" },
        { "c programs", "cookies",  "java developers", "oops"},
        { "john", "doe", "mary", "mark holmes"},
        { "hello java java", "byebye", "java again?", "java what?"},
        { "toronto", "montreal", "quebec city", "calgary"}
};
```

The main thread (i.e. the main method), not the search threads, in the application displays the following output:

```
+ search word: java
+ summary
  row 1: java, I love java
  row 2: java developers
  row 3: ***
  row 4: hello java java, java again?, java what?
  row 5: ***
+ total number of strings found: 6
```

Notes

1. Your code must work for any array size. Thus the total number of rows determines the total number of threads that will run in this application.

2. You must code at least two Java classes. The thread class should implement the Runnable interface. Each thread should have a random amount of sleeping time (between 100 ms and 2000 ms). You should use the **Random** class in Java: http://www.javapractices.com/topic/TopicAction.do?Id=62 .

3. The search word is "java". Declare it as a static variable in a Java class. The name of this variable must be in upper case.

4. Use an **ArrayList** object to store rows of strings that contain the search word.

Submission Requirements at Blackboard

Submit a) a jar file of your Java source code and b) a screenshot of the output.