# Program #2

**20 points; Due Sept. 30 (Wednesday) @ 5pm**

## Program Description

You are going to write a **multi-threaded Server** program to handle requests from multiple Client programs. The objective is to learn how to use **ServerSocket** and Java **Thread**. The Server will be "listening" on port **5764** for the connection requests from the Clients. You can run the Server and the Clients on the same machine. In this case, the Server will be a "**localhost**" with the IP Address **127.0.0.1**. You can use the Client program in Program 1 to test your Server. Once a connection request is accepted, the Server creates a new thread to handle the connection and keeps listening on port 5764 for additional connection requests. That is, your Server must be able to handle multiple connections simultaneously. The Server is also required to maintain a **log file** that records every connection.

## The Protocol

All cryptographic algorithms involve substituting one thing for another. Message in original form is called **plaintext**, and encrypted message is known as **ciphertext**. Your Server implements an old symmetric key algorithm known as the **Caesar cipher** (a cipher is a method for encrypting data.) and provide the service of encryption. Caesar cipher would work by taking each letter in the plaintext message and substituting the letter that is ***n*** letters later (allowing wraparound; that is, having the letter ***z*** followed by the letter ***a***) in the alphabet. For example, if ***n* = 3**, then the letter ***a*** in plaintext becomes ***d*** in ciphertext (and the letter ***A*** becomes ***D***.)

| plaintext | a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r | s | t | u | v | w | x | y | z |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ciphertext | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r | s | t | u | v | w | x | y | z | a | b | c |

**Polyalphabetic encryption** is to use a repeating pattern of multiple ciphers for encryption. Your Server must use **2 Caesar ciphers**, $C_1$ and $C_2$, with the repeating pattern $C_1$ $C_2$ $C_2$ $C_1$ $C_2$ for the encryption. For example, if $C_1 = 5$, $C_2 = 19$, the plaintext message "**Nice to meet you!**" results in the ciphertext "**Sbvj mt fxjm dhn!**". NOTE: the Server encrypts alphabet letters only.

| plaintext | a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r | s | t | u | v | w | x | y | z |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $C_1 = 5$ | f | g | h | i | j | k | l | m | n | o | p | q | r | s | t | u | v | w | x | y | z | a | b | c | d | e |
| $C_2 = 19$ | t | u | v | w | x | y | z | a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r | s |

The protocol is summarized as follows.

- Let Caesar ciphers $C_1 = 5$, $C_2 = 19$;
- Encrypt the message sent from the Client with the repeating pattern $C_1$ $C_2$ $C_2$ $C_1$ $C_2$ ; no substitutions on all non-alphabet characters;
- Send the ciphertext back to the Client and continuously encrypt the messages received until the Client send a "**quit**" message.
- Send a "**Good Bye!**" message back to the Client if a "**quit**" message is received, and close the connection (socket.)

## Program Requirement

1. Your Server **must not crash** under any situation, and must work with the Client program from Program 1.
2. You must try-catch everything and output appropriate error messages identifying the specific exception.
3. You MUST follow the software development ground rules from CS2430 (posted on D2L.)

4. Name the project **<your_login_name>_prog2**. Drop a copy of your project to K:\Courses\CSSE\changl\cs3830\1Dropbox by the due date, or **0 points**.
5. You MUST DEMO your program by the due date, or **0 points**. Once you demoed, the grade is final. Therefore, make sure you thoroughly test your program before you demo. Demo schedule will be announced on D2L.
6. You MUST have 3 classes in your Server program: **Server, ServerThread** and **PolyAlphabet**.
   - **Server** class should have exactly two public methods: **main()** and **run()** (you may have other private methods). The **main()** method of the Server should simply create a new **Server** object and call the run() method on it. Exceptions generated from this class must be caught and the error messages must be displayed on the Server console.

     In the **run() method**, do the following:
     (a) Create a new **ServerSocket** listening on port **5764** (refer to the socket tutorial on D2L.)
     (b) Create a new PrintWriter object with a new FileOutputStream that writes to a log file called **prog2.log**, which should be stored in your project folder. Make sure the second argument to the PrintWriter constructor is **true**. Every ServerThread will write connection information to the same log file. Each time your Server comes up, you can either overwrite or append to **prog2.log**.

     (c) Have an infinite while loop that
        i. Listens to connection requests using **.accept()** method, and once a connection is accepted, create a new instance of Socket to handle the connection.
        ii. Creates a new instance of ServerThread to handle the connection. When you're creating a new ServerThread object, use the socket object and the log file as arguments. So each ServerThread can communicate with the Client socket and write to the same log file.
        iii. Calls the **start()** method of the ServerThread object to start the thread.
   - **ServerThread** class should extend the **Thread** class and should NOT be designated as public.
     (a) Declare one Socket object, two PrintWriters (one for the socket I/O, one for the log file) object, and one BufferedReader object, outside of all methods. So they are visible in ServerThread class.
     (b) Define a constructor with the interface:
         **public ServerThread (Socket** clientSock**, PrintWriter** logfile**)**
     (c) Since ServerThread extends Thread, it MUST have a run() method, which is the method that eventually gets called after you call start() on a Thread. The **run() method** should do the following:
        i. Write a line to the log file indicating that a connection was received. You should write the *date/time* the connection was received, the remote *IP address* and the *remote port* of the connection. You may find the following methods useful for your log file: **Date().toString(), getInetAddress()** and **getPort()** of the Socket class.
        ii. Have a while loop deals with receiving/sending the messages and the encryption. You should use **PolyAlphabet** class in the loop. Your Server should loop until a "**quit**" message is received.

iii. When the Client disconnects (quit), you should print an appropriate message to the log file saying the connection on some port number is closed. DO NOT close the PrintWriter for the log file. Just close the Client socket and return from the run() method. Doing so effectively kills the ServerThread.

iv. **All output from ServerThread must go to the log file. No output may be sent to System.out.**

- **PolyAlphabet** class implements the polyalphabetic encryption using 2 Caesar ciphers $\boldsymbol{C_1}$ and $\boldsymbol{C_2}$ with the repeating pattern $\boldsymbol{C_1\ C_2\ C_2\ C_1\ C_2}$.

7. Your Server.java and ServerThread.java should look something like these:

```
import java.io.*;
import java.net.*;
import java.util.*;
class Server
{
   public static void main(String argv[])
   {
      ...
   }
   public void run()
   {
      ...

   }
}

class ServerThread extends Thread
{
   ...
   public ServerThread (Socket clientSock, PrintWriter logfile) //constructor
   {
      ...
   }
   public void run()
   {
      ...
   }
}
```

8. Log file entries may look something like these:

```
Got a connection: Fri Sep 05 13:40:16 CDT 2014  /137.104.121.250  Port: 9312
Connection closed. Port: 9312
Got a connection: Mon Sep 15 13:32:41 CDT 2014  /137.104.121.234  Port: 11132
ServerThread Exception: java.net.SocketException: Connection reset
```

9. You can run and test your Server and Client on one machine. To do so, bring up your Server first, and run your Clients with "**localhost**" (**127.0.0.1**) and port **5764**.

10. If your programs work on the same machine, run and test your Server and Client with different machines in the labs. You could use "**ipconfig**" command at the command prompt under Windows to find out the IP address of the machine running your Server program. Run your Client program with that IP address to connect to your Server.

**Program Grading**

| **Exceptions/Violations** | **Each Offense** | **Max Off** |
|---|---|---|
| Program not running | 20 | 20 |
| Cannot handle multiple connections | 20 | 20 |
| Encryption logic errors | 2 | 10 |
| Missing the log file | 5 | 5 |
| Improper implementation on the Protocol | 2 | 6 |
| Improper handling try/catch exceptions | 0.5 | 2 |
| Not recording connections properly in the log file | 0.5 | 2 |
| Not complying CS2430 programming ground rules | 0.5 | 5 |