CS 251: Lab 10: Java: Eclipse IDE, Sockets, RSA (Outlab)

- Handed out: 10/10 Due: 10/13 11pm
- Please write (only if true) the honor code. If you used any source (person or thing) explicitly state it. You can find the honor code on the web page.

Overview

In this lab we will use Eclipse, a very popular integrated development environment (IDE). An IDE normally consists of a source code editor, build automation tools and a debugger. Most modern IDEs have intelligent code completion.

Using Eclipse, we will build a peer to peer chat service.

Pre-tasks

1. Install Eclipse (It's a very large download, 270MB for the J2EE version, you want the smaller 165M standard edition. You also want to do peer sharing of this to avoid insane net bandwidth usage). Eclipse comes with an extensive tutorial. You should be using those instead of the ones on the net.

   Still, if you don't believe us, these resources will help you get started with Eclipse :

   - `https://eclipse.org/users/`
   - `https://www.youtube.com/watch?v=sbo-D3P6GNg`

   An IDE can be daunting at the start (i.e., steep learning curve), but once you master it, you can enjoy many of the benefits.
2. For sockets, look up `https://docs.oracle.com/javase/tutorial/networking/sockets/` to learn about Java Sockets.
3. **For inlab, read the basic SWING documentation**

The Tasks

1. `[Warmup]` We are just going to take some existing code and import it to Eclipse.
   (a) Download `mathops.zip` (called archive in Eclipse terminology) from the usual resources directory.
   (b) Start Eclipse, go to `File->Import->General->Existing Projects Into Workspace`
   (c) Select the archive file option, and browse to the downloaded zip file and click Finish.
   (d) This will create a project named MathOps in your workspace (Take care that you don't have any project by that name in your workspace from earlier).
   (e) Now you can see a cross mark besides the icon of MathOps in Project Explorer section of Eclipse. This means there is a compile error in project. Find the error and rectify it. Once done, try to use the debugger to find another logical error. (The goal here is to get you used to the IDE debugger. You can always find the error by manually inspecting the code, or inserting `println` but then why are you using an IDE?)

   (f) After fixing errors, run the project and record the output of the program. **Mention the error and output of file in your readme**.

   (g) Export your project as an archive to `myMathOps.zip`. This file should work "out of the box" when the grading TAs imports this file into their Eclipse workspace. However make sure that you don't submit anything extra; bin subdirectory is not to be submitted. The easiest way to check if all is well, is to get your partner to perform the steps the TA would.

2. `[Ghosts from the Past]` In the previous step we imported a file that has been earlier *deployed* by Eclipse, so the import was easy. But how do you take a file that was not created/developed in Eclipse? Here we will be reorganizing Q2 of outlab from Lab 08.

   Alert: Names are very important in this task. Also you might find that your earlier "working" program is suddenly not compiling – it's Eclipse trying to help you.

   (a) Create a new Java project by name Lab08Q2

   (b) Now create a package in src by name `myLibrary`

   (c) The package must have three classes namely Book, Reader and Writer. These must be the same files you created in Lab08. Add these classes into your myLibrary package.

   (d) Create a ReaderWriter class in the default package. (This is again same class as the one submitted.) Note that Eclipse discourages default packages, but go ahead and do it.

   (e) Now ensure that when you press run, everything works fine. For this you may have to do add some `import` statements. Try not to change your code except for adding these statements for your project to work. (You should see some shortcuts from Eclipse while typing).

   (f) Export your project to `myGhost.zip`. This file should work "out of the box" when the grading TAs imports this file into their Eclipse workspace. However make sure that the bin subdirectory, for example, is not submitted. They will be looking out for the `myLibrary` package.

3. `[Chat Service]`

   The chat service will have two counterparts, a client and a server. Your program should prompt whether it has to be started as a client or a server. For this part, the server runs on the PC of person with lowest roll number say X. And client runs on the PC of person with middle roll number say Y.

   (a) Initially a server runs and listens to port **5123** for a client request.

   (b) When invoked as a client, Y takes the IP address of the server (which needless to say, is running on X) from the standard input. It then sends a prompt message: "Hey! Want to chat?" to the server on X and waits for random time between 5 to 10 seconds.

   (c) If no answer is received, an appropriate timeout message is displayed and the process is terminated.

   (d) If a proper response is received, (we will be using "Sure!"), a chat session is initiated. Then Y and X will be able to send messages to each other in packets. The length of the message (actual content) should be limited to 140 characters. Truncate if necessary. Along with the message, each packet contains a date and time-stamp. These are not printed for the end user, but available to the client (and not counted in the 140 limit).

   (e) A special string "End Chat" will be sent by the client on Y to end the session.

   (f) At all times, a unoccupied server should be listening on the port, waiting for a client to connect.

(g) Once you have completed the program, you will either complete Task 4 (up ahead), or bail out. If you want to bail out at this stage, export your project to the File System, and **not as an archive**. Your TA will not use Eclipse to grade this part of the assignment. You should export only the source files. You should also externally create a makefile as mentioned below in the submission instructions.

Check the files `serverlog.txt` and `clientlog.txt` in the resources folder to see how your chat should look like. These files are examples.

You can assume that the server will talk to only one client at a time. You will have to design the following classes in Java:

(a) `JMessage` This will contain only the string message.

(b) `JPacket` This will contain the entire packet: a time stamp and the string message.

(c) `JChatComm`
- method `sendMessage` - send a message to a particular ip
- method `receiveMessage` - receive a message and output it to screen
- method `endChat`

(d) `JClient`
- method `callServer` - to call a server and initiate a chat

(e) `JServer`
- `constructor` to start listening on a port
- method `acceptConnection`

You are free to decide relationship between these classes and their data members. Make sure only members/methods needed to produce output are public, rest everything should be private. You are free to create more classes if you want, but you must create the ones asked above.

4. `[RSA Encryption]` When Y is talking to X, the person with highest roll number, say Z, is feeling sad and left out. And hence he will try to snoop into the conversation which is going on the public network (between X and Y). To prevent snooping attacks, we will introduce RSA encryption in our chat service. In RSA, even if an intruder gets the entire message, she is unable to decode the message "no matter what".

In RSA crypto-system, the encryption key is public and differs from the decryption key which is kept secret. A user of RSA creates and then publishes a public key based on two large prime numbers, along with an auxiliary value. The prime numbers must be kept secret because revealing this will imply that the private key is compromised. Anyone can use the public key to encrypt a message, but with currently published methods, if the public key is large enough, only someone with knowledge of the two prime numbers can feasibly decode the message. (For those who missed the trailer on Saturday, (and for others too!) this video might help `https://www.youtube.com/watch?v=Z8M2BTscoD4` you with RSA.)

Important note: Generally a programmer does not implement any type of cryptography by himself; rather library func is used. Modify your code for the above part as follows

Note: This task must be started ONLY after the previous task has been completed.

(a) When the Java program is started, it constructs a random public key and a private key (both client and server) as per the RSA scheme. The public key is communicated as part of the chat initiation as described below.

(b) When invoked as a client, the program requests to the IP address of the server, sends a prompt message “Hey! Want to chat?. My public key is : `5029528498025999493`” to the server and waits for a random time between 5 to 10 seconds. (TA checking note: Ideally we want different groups to have different public keys.)

(c) If a proper response is received from the server, (“Sure! My public key is : `37528593287538`”), a chat session is initiated. (TA checking note: Ideally we want different groups to have different public keys.)

(d) Then the client and server will send only encrypted messages to each other in packets. The length of the message (the plain-text message) should be limited to 140 characters. Truncate if necessary. Along with the message, each packet contains a date and time-stamp.

(e) `[Optional]` The final End Chat message should be followed by a digital signature: “Signed: Client Group Id 0X” This last message should be a digitally signed so that the server can verify the authenticity of the chatting client so that logs can be maintained.

(f) Once you have completed the program, export your project to the File System, not as an archive. Your TA will not use Eclipse to grade this part of the assignment. You should export only the source files. You should also externally create a makefile as mentioned below. This is a cumulative submission,i.e., if you submit this part, then do not separately submit Task 3.

`[Hint]`: Use BigInteger.probablePrime method to create large random primes.

Important Notes for 3 and 4: `[Project Design]`

- All classes except the main method should go into their own separate files and form the part of a package named “cs251lab10package”.
- The file with the main method should go into a file called “lab10Main.java”
- Put all your source files in a folder named “src”. Write a makefile with a target `compile` that will perform the compilation. All binaries should go to a separate folder named “bin”. Also, create a target `clean` to clean all generated files.
- Create a target in your makefile named `startClient` and a target named `startServer` which will run your program as client and a server respectively. Your client should then request for the IP address of the server.
- In your README file explain the relationship between the classes and the encryption algorithm in brief.

What to Submit

1. Add the src folder of ChatService and Makefile to folder by name ChatSrvc.
2. Include the zip files of the other tasks in separate folders called q1 and q2 (both of which will contain zip files). Don’t call it Question 1 or Question-1 or ...
3. Submit the folders ChatSrvc, q1 and q2 and a readme. Don’t forget your honour code.
4. Put everything in a folder. The folder and its compressed version should both be named `lab10_groupXY_final`. Hence, you submit a tar.gz named `lab10_group07_final.tar.gz` if your group number is 7.

`..........................   Gooooooooooooooooooood Luck       ..................`