CS416 Project 2: Basic File Server and Client

You can team up with another student or work alone. Each team can have no more than two students.

Objectives:

1. Gain deeper understanding of the server-client architecture.
2. Reinforce the concepts of TCP logical connection and the byte stream model.
3. Learn and practice Java socket programming.
4. Design and implement an application-level file transfer protocol

When it is due:

February 25th, Friday, at 11:59pm.

What to submit:

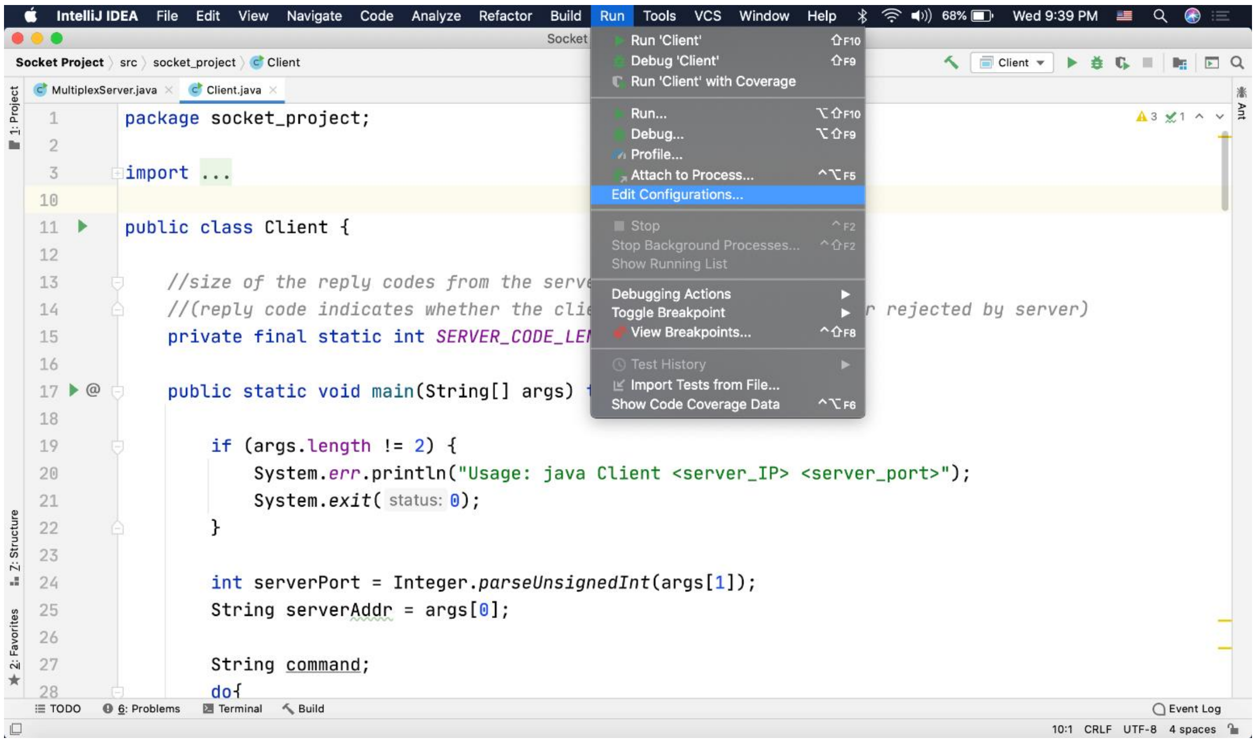
1. Please submit the completed **MultiplexServer.java** and **Client.java** files in Canvas.
2. *After the submission*, please schedule a time with the instructor to demo your project. The demo should only take 10-15 minutes. Please note that the demo will be done with the server and the client each running on a separate host (if you work alone and only have one laptop, the instructor will provide an additional laptop for the demo.)
3. The demo can be done after the submission deadline. The deadline only applies to the submission of the Java source files.

For a two-person team, you will just need to submit *one* copy (either member submits in Canvas). Please be sure to add a submission note stating it is a team submission with the names of both team members. Additionally, both team members must attend the demo session.

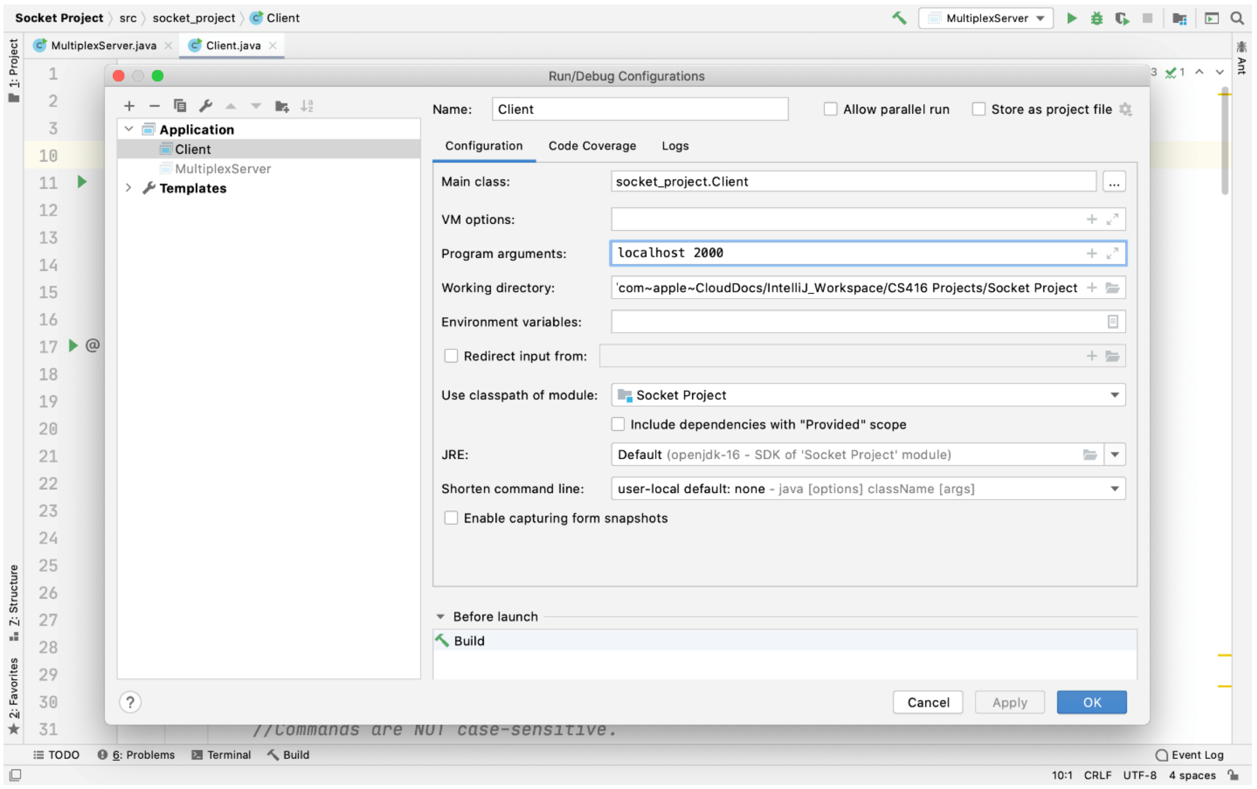
Instructions:

1. For this project you will need to work on two Java programs (server and client). We highly recommend that you use a Java IDE, such as the IntelliJ IDEA (the free Community version).
2. You are provided with the *incomplete* source code of the server and the client. Your task is to complete both of them to add support for the following commands:
    a. "D" – client requests to delete a file on the server (client specifies the name of the file)
    b. "G" – client requests to download a file from the server (client specifies the name of the file)
    c. "R"– client requests to rename a file (client specifies the original file name and the new file name)

3. Two additional commends, “L” and “Q”, have been implemented already.
    a. “L” – client requests a list of available file names and their sizes on the server
    b. “Q” – client quits (in this case, nothing is sent to the server)
4. For each client request, server will send back a *reply code*, along with any actual data that were requested by the client.
    a. The reply code is either “S” or “F” depending on whether the requested operation can be accepted or not. An unacceptable request, as an example, may be a Delete request with a file name that does not actually exist on server.
5. We provided the *complete* implementation of the “L” command on both the client and the server. Please be sure to review our implementation and you can use it as a template for implementing the other commands.
6. You will not need to modify any existing code. You will just need to add implementation to the *switch* statement on both the client and the server.
7. The **client** program requires two command-line arguments to run: the first one is the IP address of the server (if the server runs on the same machine as the client, use *localhost*), and the second argument is the port number of the server (use *2000*, as the server is hardcoded with that port number).
   If you use IntelliJ, you can specify those two arguments in the “Program arguments” box, which is accessed by clicking the “Edit Configurations…” option under the “Run” tab (please refer to the screenshot below).



```
package socket_project;

import ...

public class Client {

    //size of the reply codes from the serve
    //(reply code indicates whether the clie                          r rejected by server)
    private final static int SERVER_CODE_LE

    public static void main(String[] args) 

        if (args.length != 2) {
            System.err.println("Usage: java Client <server_IP> <server_port>");
            System.exit( status: 0);
        }

        int serverPort = Integer.parseUnsignedInt(args[1]);
        String serverAddr = args[0];

        String command;
        do{
```



Name: Client

Allow parallel run ☐ Store as project file ☐

Main class: socket_project.Client

VM options:

Program arguments: localhost 2000

Working directory: 'com~apple~CloudDocs/IntelliJ_Workspace/CS416 Projects/Socket Project

Environment variables:

☐ Redirect input from:

Use classpath of module: Socket Project

☐ Include dependencies with "Provided" scope

JRE: Default (openjdk-16 - SDK of 'Socket Project' module)

Shorten command line: user-local default: none - java [options] className [args]

☐ Enable capturing form snapshots

```
//Commands are NOT case-sensitive.
```

8. Make sure you run the server program first before you start the client program, as the client will try to connect to the server. The server does NOT need any command-line argument, and it runs forever (the only way to stop it is to manually kill it, e.g., by clicking the red “stop” button in IntelliJ).

Grading:

This project carries 100 points. You will receive 50 points each for correctly implementing the client and the server. You will receive partial credit if your implementation is partially correct.