EE 579 Diagnostic Exam Take-Home Exam

Assigned on: Tuesday, Jan 14, 2014

Due on: Tuesday, Jan 21, 2014

INTRODUCTION

This exam consists of a mini-project, which will help you get started with the basics of Java. If you do not know Java, you should start learning it now. However, it may be emphasized that our goal is to not make you a master of any specific tool (for example, Java), but to get you adept in learning how to use appropriate tools.

What you will learn by doing this project:

1. Understand Java classes, and possibly concepts like Abstraction, inheritance etc.
2. Understand Java packages.
3. Develop coding etiquettes.
4. Get started with socket programming in Java.

BEFORE YOU GET STARTED

Get going on the following steps:

1. Install Eclipse IDE and get familiar. This is not required, but Eclipse is a great tool to write Java programs. You might find it especially useful if you write Android programs later. Start with a simple program and learn how to use the debug perspective of Eclipse.
2. Read Oracle code conventions at [1] and use it in your programs. Most importantly, use 2 space indentation, no tab, no tailing space and 80 char line width. Edit your Eclipse preferences to make this default. Also, learn to name variables. As an example, suppose you have a Boolean variable that specifies whether you need to perform light decoding or full, you may come up with the following:

   ```
   boolean lightdecoder = false;
   ```

   This has wrong capitalization and instead lightDecoder is preferred. In fact, a better variable name would be

   ```
   boolean doLightDecode = false;
   ```

   Which conveys the functionality.

3. Learn to use packages. Eclipse will automatically create new directories and place your classes in those directories as per the package definition, but keep an

eye on where it is placed in your file system.

PROJECT DESCRIPTION

In this project, you are required to build a server program that will listen to a specific port; and a client that will connect to the server, handshake, request a list of files, display it to the user and prompt which one to download, and send a download request to the server. Note that there is no need to transfer any file. Once the server gets a request to download a file, it will respond with the content of the requested file (a few sentences).

All the communication between the server and the client must take place in a common language that both the server and the client will understand.

Communication will take place by exchanging packets with a predefined structure. Each packet should adhere to the following format:

API_TYPE (4)	TOTAL_LEN (4)	MSG_LEN (4)	Message (x)

The bracketed terms represent the size in bytes of each of the field. The message can have variable length (upto a maximum of 10000), and its length x will be stored in the third field “MSG_LEN”. We will fix the API_TYPE to a number 12 for this project. You are required to use four bytes to store the number 12 and take care of the network byte order (do not store as a string). The TOTAL_LEN will be 4 + 4 + 4 + x.

Message should be comma delimited sequence of strings (here you can store everything including numbers as strings).

Message type, message1, message2, ..

A client can send the following formats:

1. HELLO: Hello message
2. LIST: List all files.
3. GET: Download a particular file (specified by its index).
4. END: terminate connection

Except the GET message, all other messages contain only the message type.

For example a HELLO packet must look like this:

12	17	5	HELLO

(If the message types are HELLO or LIST or END, make sure the MSG_LEN is 5, 4 or 4 respectively).

Once the server gets a message, it will respond in the following way, using the same packet structure defined above:

For example, once the server receives a LIST message, it could send back

```
LIST, 0, file0, 1, file1, 2, file2
```

or

```
LIST, 34, Somename.txt, 67, Another.dat
```

When the client gets this, it has to display these files to the user.

1. Somename.txt
2. Another.dat

If the user now chooses to get Another.dat (by pressing 2), the client must send a GET message to the server with the index 67 as below:

```
GET, 67
```

The server, upon receiving this message, will send back some message (which is supposed to be the content of the file), which the user must display to the user.

```
GET, 67, Another.dat, A small piece of text
```

So the user will display “A small piece of text” to the user. If the server cannot determine what the message type is, it will just send an INVALID message.

IMPLEMENTATION DETAILS

Your Server and Client classes should be in a package called

```
edu.usc.anrg.ee579.diagnostic
```

You should also make use of at least one another package, for example, say

```
edu.usc.anrg.ee579.diagnostic.protocol
```

All classes with main() that you use to test should be in

```
edu.usc.anrg.ee579.diagnostic.test
```

Keep the server IP and the port as variables which can be changed, with default values pointing to the localhost and port number 9777.

You are free to develop the way your server and client work, but they should work together and not cause problems.

NOTES

1. This exam is meant to act as a diagnostic to judge your competence for the course. It is worth 5 percent of the coursework.
2. Although you may consult passive references (books/articles), you may not communicate with or seek the assistance of others. Each student is required to do this exam on their own. Try not to lift code from the Internet, but if you must use something, give a link in the code as a comment.
3. Place clear comments, again, following the convention from [1].
4. Each method must be preceded by a short description of what it does.
5. If you have never encountered socket programming before, there are tons of tutorials on the web that can help you.
6. You must upload the source files as a zip file to the Blackboard dropbox by 11.55 pm on Wed, Jan 18. When creating the zip file of the source files, start from the directory 'edu'. You will not be able to upload the file after the deadline. Students who have not registered can email the file to deora@usc.edu
7. Don't wait to start on this exam! It is likely to be quite time-consuming.
8. Any further questions regarding the exam must be communicated to the TA by email.

References/Links

1. Oracle Code Conventions: http://www.oracle.com/technetwork/java/codeconvtoc-136057.html
2. Eclipse IDE: http://www.eclipse.org/
3. Simple Java socket program example: http://docs.oracle.com/javase/tutorial/networking/ sockets/clientServer.html