GALWAY-MAYO INSTITUTE OF TECHNOLOGY

Department of Computer Science & Applied Physics

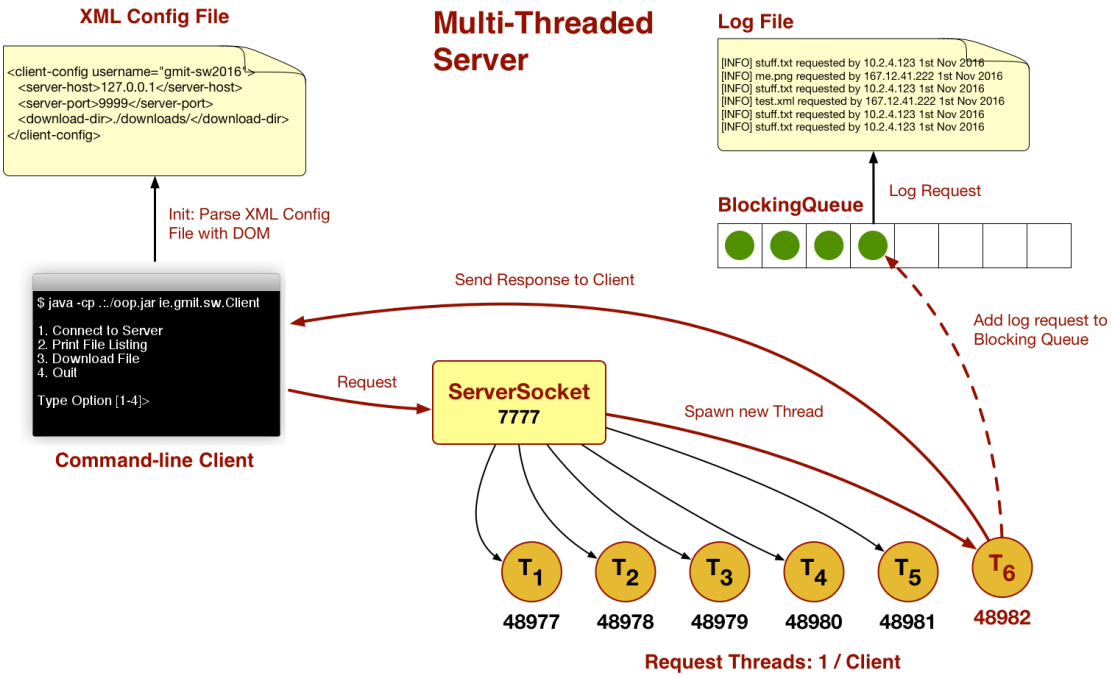
Object-Oriented Programming (2016)
ASSIGNMENT DESCRIPTION & SCHEDULE

A Multi-Threaded File Server

Note: This assignment will constitute 50% of the total marks for this module.

1. Overview

You are required to implement a ***multi-threaded file server and logging*** application that allows a client application to download files using a set of options presented in a terminal user interface. An overview of the application components is depicted below:



2. Requirements

- The client application should present a command-line user interface when started from a terminal window as follows: ***java -cp .:./oop.jar ie.gmit.sw.Client***, where **oop.jar** is the name of the JAR archive containing the full set of classes (client and server) for the project. The following options should be offered to a user:

1. Connect to Server
2. Print File Listing
3. Download File
4. Quit

Type Option [1-4]>

Use a *java.util.Scanner* object to read in console input from the user. The command-line menu should use a ***while*** loop to keep the client application alive between interactions. Selecting option (1) should open a socket connection to the server, option (2) should query the server and display the list of files that are available for download, option (3) should prompt the user to specify a file from this list and then download the file from the server. Option (4) should cause the ***while*** loop to exit.

- When the client application is started, it should parse an XML file using the **Document Object Model (DOM)** and extract from it the set of values required to interact with the remote server. The XML document should be structured as follows:

```
<client-config username="gmit-sw2016">
    <server-host>127.0.0.1</server-host>
    <server-port>7777</server-port>
    <download-dir>./downloads/</download-dir>
</client-config>
```

where the attribute ***username*** is the avatar name used when interacting with the server application and the elements ***server-host*** and ***server-port*** are the IP address of the remote file server and the socket port number to connect through (use the port **7777** as a default). The element ***download-dir*** is the local directory used by the client application to stored downloaded files from the server.

- The server application should be started with the following command and be packaged in the same JAR achive (oop.jar) as the client:

java -cp .:./oop.jar ie.gmit.sw.Server 7777 /path/to/myfiles

where *7777* is a port number and ***/path/to/myfiles*** is the relative or absolute path to the directory containing the set of files to download. The server should execute a **fully threaded** *ServerSocket* on the specified port (7777), i.e. each client socket must be executed in its own thread on the server when the option (1) command is executed by a client.

- Every request received by the server should be added to a blocking queue by the thread and logged in a text file using the following format: [INFO | ERROR | WARNING] <command> requested by <client ip address> at <date time>, e.g.:

[INFO] Listing requested by 127.0.0.1 at 12:38pm on 1st November 2016

A single thread should be used to remove items from the ***blocking queue*** (i.e. many producers and one consumer). Note that a blocking queue will not know when to stop waiting… A common technique is to use a "poison" object to tell the queue to die.

- Use the ***java.io.File*** class on the server to get a directory listing by calling the method *list()* or *listFiles()* on the directory specified as ***/path/to/myfiles*** (note that directories and files are treated as the same entities). Use the ***java.io.FileWriter*** class to log each request to a file.

- The whole point of this assignment is for you to demonstrate an understanding of the principles of object-oriented design by using abstraction, encapsulation, composition, inheritance and polymorphism WELL throughout the application. Hence, ***you are also required to provide a UML diagram of your design and to JavaDoc your code***. Please pay particular attention to how your application must be packaged and submitted. Marks will be deducted if you deviate from the requirements.

3. Deployment and Submission

- ***The project must be submitted by midnight on Sunday 8th January 2017*** using both Moodle and GitHub.

GitHub:

- Submit the HTTPS clone URL, e.g. https://github.com/myaccount/my-repo.git of the public repository for your project. All your source code should be available at the GitHub URL. *You should try to use GitHub while developing your software and not just push changes at the end.*

Moodle

- The project must be submitted as a Zip archive ***(not a rar or WinRar file)*** using the Moodle upload utility. You can find the area to upload the project under the "A Multi-threaded File Server (50%) Assignment Upload" heading of Moodle.
- The name of the Zip archive should be <*id*>.zip where <*id*> is your GMIT student number.
- The Zip archive should have the following structure (do NOT submit the assignment as an Eclipse project):

Marks	Category
oop.jar	A Java Archive containing your API and runner classes for both the client and server with a main() method. You can create the JAR file using Ant or with the following command from inside the "bin" folder of the Eclipse project: **jar –cf oop.jar ***
src	A directory that contains the packaged source code for your application.
README.txt	Contains a description of the application, extra functionality added and the steps required to run the application.
design.png	A UML diagram of your API design. Your UML diagram should only show the relationships between the key classes in your design. Do not show methods or variables in your class diagram.
docs	A directory containing the JavaDocs for your application.

4. Marking Scheme

Marks for the project will be applied using the following criteria:

Marks	Category
(50%)	Robustness
(10%)	Cohesion
(10%)	Coupling
(10%)	JavaDocs and UML Diagram
(10%)	Packaging & Distribution (GitHub and Moodle)
(10%)	Documented (and relevant) extras.

You should treat this assignment as a project specification. Any deviation from the requirements will result in a loss of marks. Each of the categories above will be scored using the following criteria:

- 0–30% Not delivering on basic expectation
- 40-50% Meeting basic expectation
- 60–70% Tending to exceed expectation
- 80-90% Exceeding expectations
- 90-100% Exemplary