Assigned: Tuesday, April 2
Due: Thursday, April 11

Notes:

- A working solution will be demonstrated in class (April 2).
- Socket programming in Java will be reviewed in class (April 2).

You may use either NetBeans or IntelliJ for this assignment. The entire project folder needs to be submitted so that I can open the project in the IDE. You may want to review the Java style guide for the course.

Name of project to be created: *SmithThreadPool* with your last name in place of *Smith*.

An echo server is a server that echoes back whatever it receives from the client. For this project you will create a multithreaded echo server in Java that uses a thread pool to limit the number of active threads. You will need to implement the following four classes:

1. ***EchoServer***. The server opens a server socket and listens (in an infinite loop) for connections. When it accepts a new connection, the server creates a *Connection* object (described below) and adds it to the thread pool.

2. ***Connection***. The constructor initializes an instance variable with a reference to a socket. This class implements the *Runnable* interface, and the run method has a loop that reads a line of text from the client and then writes the same line back to the client. You may need to flush the output stream after writing.

3. ***ThreadPool***. You will need two inner classes:

   - ***WorkQueue*** stores a linked list of runnable tasks. Use the LinkedList class for this. Note that this class is not thread safe, which is why you must synchronize access to it (see what happens if you do not). You need one method to add a task to the list and another to get a task. A thread will block if it tries to get a task when none is available.

   - ***WorkerThread*** implements the *Runnable* interface. Its *run* method simply tries to get a task from the queue and run it.

   The *ThreadPool* constructor creates an array of threads and instantiates the work queue. It then fills the array with *WorkerThread* objects and starts them. The size of the array should be determined by a static constant (initialized to 3 for testing purposes). The only other method needed for your *ThreadPool* class is one that adds a given task to the queue.

4. ***EchoClient.*** The client tries to open a connection on the local host. It reads a line of text from the keyboard, writes it to the server and then writes the server's response to the console. This continues until the user enters a single period, which terminates the connection.

Your project will include two packages named *threadpool* and *threadpool2*. Each of these will contain an implementation of the four classes described above. Three of those classes (*EchoServer*, *EchoClient*, and *Connection*) will be identical in the two packages. The only difference will be in the *ThreadPool* class:

- *ThreadPool* in the first package uses synchronized methods to synchronize access to the list.
- *ThreadPool* in the second package use *ReentrantLock* and *Condition* objects for synchronization.

Execution snapshot for server:

```
Server started.
Thread pool created.
Awaiting clients.
```

Execution snapshot for a client (user input shown in **bold green**):

```
Echo client starting.
Echo client connected.
Client: LAY DOWN ALL THOUGHTS, SURRENDER TO THE VOID
Server: LAY DOWN ALL THOUGHTS, SURRENDER TO THE VOID
Client: TANGERINE TREES AND MARMALADE SKIES
Server: TANGERINE TREES AND MARMALADE SKIES
Client: WORDS ARE FLOWING OUT LIKE ENDLESS RAIN INTO A PAPER CUP
Server: WORDS ARE FLOWING OUT LIKE ENDLESS RAIN INTO A PAPER CUP
Client: .
Client done.
```

When testing, you should first start the server and then start multiple clients. Try typing lines of text into the different client consoles. Only three clients should be active; when one is terminated then the output for one of the waiting clients should appear in its console.