Assignment 1

Formalities

This assignment may be solved in groups of up to 3 members. Each member must hand in themselves. Attach a file naming the group members.

Deadline is Friday the 30th of November.

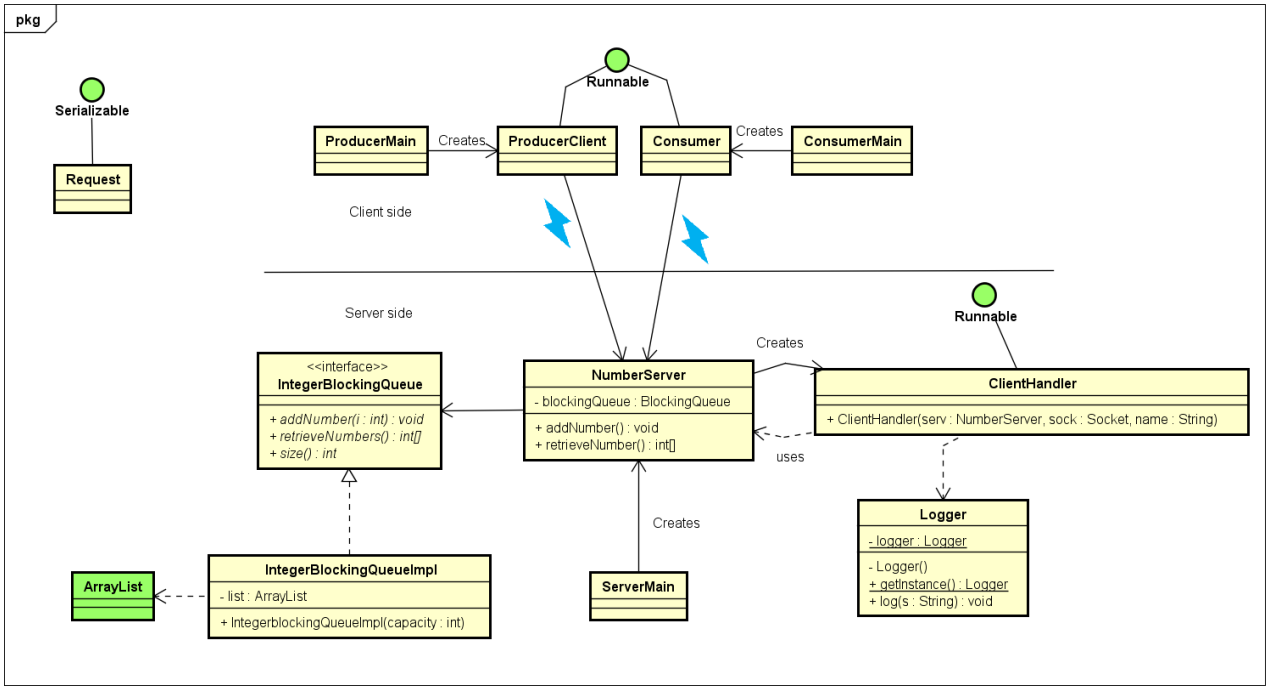
This assignment must be handed in an accepted, for you to go to the exam.

Topics for this assignment:

- Sockets
- Threads
- Producer/consumer
- Blocking queue
- Singleton
- Adapter

In this assignment you're going to implement a client server system using sockets. The server will contain a queue of integers. One type of client will put integers into the queue, another type of client will take numbers out of the queue.
Each client will be handled by a separate thread, so the queue needs to be a blocking queue.
Each thread will on the server will also output information through a shared logger instance. Many threads need to write to the log, and they should all use the same, so here we use the singleton pattern.

Here's a rough UML diagram:



The green classes are from Java.

You don't have to follow this diagram exactly, but there are some requirements.

1. The NumberServer acts as the server. It should accept Socket connections from clients. Each connection will be handled in a separate thread: ClientHandler. It has a field variable of type IntegerBlockingQueue, and two methods add/retrieve, to interact with the queue.
2. The ClientHandler is a thread, it implements Runnable. It is responsible for reading from and writing to the clients. This means it needs to be able to handle different types of requests. You may want to send a Message/Request object with variables to determine what kind of request it is. Every use of the Logger singleton happens from this class.
   Give each thread a name, so you can output this later to the console.
3. The Logger is a *singleton*. The log method will just print out the String *s* to the console. You must make this thread safe.
4. The IntegerBlockingQueueImpl class is a Blocking Queue implemented by using an arrayList. See the slides from the second threads session (Threads Monitor), the example about the bar/beers/waiters/customers. The capacity limit of the queue is 20. The class implements an IntegerBlockQueue interface. This here is the adapter. Your system needs a BlockingQueue working with integers, so you're adapting the java.util.ArrayList to work as this BlockingQueue.
5. The ProducerMain and ConsumerMain starts each client, respectively, and runs the thread.
6. One type of client, ProducerClient, will add a random number to the BlockingQueue over and over at a regular interval, e.g. every 100ms. It implements Runnable, so this is a Thread. In the run method you should create a connection to the server, and then in a for-loop add numbers to the server. You can use the following code

```
Random r = new Random();
int i = r.nextInt(100);
```

   It will create a random number between 0 and 99. Send this number to the server through the ClientHandler. The ClientHandler has a reference to the NumberServer, and can call the add/retrieve

methods on the NumberServer. The NumberServer will in turn then call the methods on the BlockingQueue. In the ClientHandler get a reference to the Logger singleton, and print out which number was retrieved.

7. The ConsumerClient will first retrieve the next 3 integers from the queue. It will log out which three numbers through the Logger singleton.
   Then on the client side add the first two, and multiply by the third: (a + b) * c.
   The ConsumerClient will then send the result back to the Server, it will retrieve an instance of the Logger singleton, and log out the result.
   This is also performed in a while(true) loop, at a regular interval.

Implementation suggestions

This may seem like a fairly large task, so you're going to have to divide the problem into smaller, more manageable problems. Think about the Work Breakdown Structure.

Recommend something along the following lines:

- First get a simple client-server connection started using sockets. The client just sends a String, then the server prints it out.
- Then expand, so the client connection is handled in a separate thread, so you now can handle multiple clients.
- Have multiple clients of the same type, ProducerClient, sending data to the server, which just prints it out.
- Create the Logger singleton, and use it in the ClientHandler to print stuff.
- Create the interface IntegerBlockingQueue, then the IntegerBlockingQueueImpl class. This just has a field variable of type ArrayList<Integer>.
  In the add method of IntegerBlockingQueueImpl, you must check the size. It must be at most 20. Look through the slides from the second Thread session to refresh blocking queues.
- Modify everything, so that whatever int is sent from the client is now inserted into the blocking queue.
- Create the last client.