Thread Programming and Non-blocking Java Servers

General part

Explain about Thread Programming including:

- When and why we will use Threads in our programs?
  We use threads to make our java applicatopn faster, as it makes us able to do multiple things at the same time. If you have GUI it also helps that you can look for and handle input while the program is still going. Smooth! Some super simple programs will not benefit from threads.

- Explain about the Race Condition Problem and ways to solve it in Java
  The race condition is a problem when multiple classes are trying to get to the same data at the same time. This can make things go wrong, so we lock the data so we dont use the same thing two places at the same time.

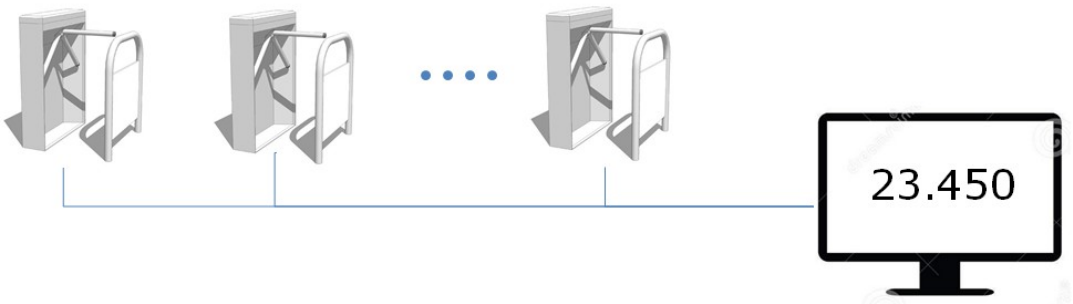
- Explain how we can write reusable non-blocking Java Controls using Threads
  Control windows

- Explain about deadlocks, how to detect them and ways to solve the Deadlock Problem
  If program 1 requests a recourse a, and program 2 requests recourse b and they both recieve it.
  Then program 1 requests recouse b, but that is being used in program 2.
  Then if program 2 requests recourse a, it wont reciev it, as its being used in program 1.

  This is a deadlock, both programs will be stuck like this.
  We can solve this by makeing things atomic so t

Practical part

In this exercise we will simulate a large football stadium with many turnstiles that each updates a shared counter, for each turn on the turnstile.



The turnstiles should use a Network Connection and TCP to update the shared counter.

Implement a Java Based server that can handle n-turnstile clients without any lost updates.

1. Design a TCP server and a simple protocol where each turnstile initially reports that it is a Turnstile (to distinguish from Monitor-Clients, see next step), its id (turnstile1-turnstile-n) and then reports an increment for each spectator that passes the turnstile
2. Identify potential Race Condition Problems and handle the problem(s).
3. For this exercise you don't have to implement the turnstile-clients. Use a few Telnet Clients to simulate the turnstiles.
4. Extend the system, so A Monitor-Client can request the current total amount of spectators
5. Change the example to make it possible to see the count from each turnstile
6. *If you have time-1: Deploy your server to DigitalOcean and demonstrate using this remote server*
7. *If you have time-2: Write a simple client (Swing or Web-based) that can show the total amount of spectators*

Hints:

For part-1 (the protocol) it could be as simple as you have to send "things" in this order:

- **TURNSTYLE** (to signal that this client is a turnstile, this is sent only once)
- **T-1** (the id of this turnstyle, this is sent only once)
- **Count** (the value to update the shared counter with, sent as many times a necessary)

Feel free to come up with an alternative protocol.

Remember: For the final exam the grades for passed lies between 02 - 12.

You are not ALL expected to be able to complete all steps for an Exam exercise. For this exercise, try as a minimum, to complete step 1-3.