Exercises Threads-1 Spring 2016

Fork (or clone, and delete the .git folder) this repository:
https://github.com/Lars-m/threadExDay1.git to get the start code for ex-3,4,5.

Exercise 1 (create, start and end threads)

Create a program that starts 3 different parallel threads.

- **task1** : Compute and print the sum of all numbers from 1 to 1 billion
- **task2** : Print the numbers from 1 to 5. Pause for 2 seconds between each print.
- **task3** : Print all numbers from 10 and up. Pause for 3 seconds between each print.

The program should stop task3 after 10 seconds.

Hint: For the sum in task-one, use the a long data type
Hint2: Let the main thread sleep for 10 seconds after starting task 3
Hint3: Use a Boolean value in the loop in task-3 to terminate task3 (let the main thread change the value of the boolean value.

Exercise 2 (race condition)

a)
The method `next()` in the class Even should always return an even number (see code snippet below). Implement a program that demonstrates that this is not always true in a multithreaded program.

```
public class Even
{
  private int n = 0;
  public int next()
  {
    n++;
    n++;
    return n;
  }
}
```

Hint: Create two threads, which both should call the next() method on the same Even object and test if the return value is equal.

- Explain what happens?
- How common is the problem?

b) Introduce synchronization, so the execution of the method is atomic

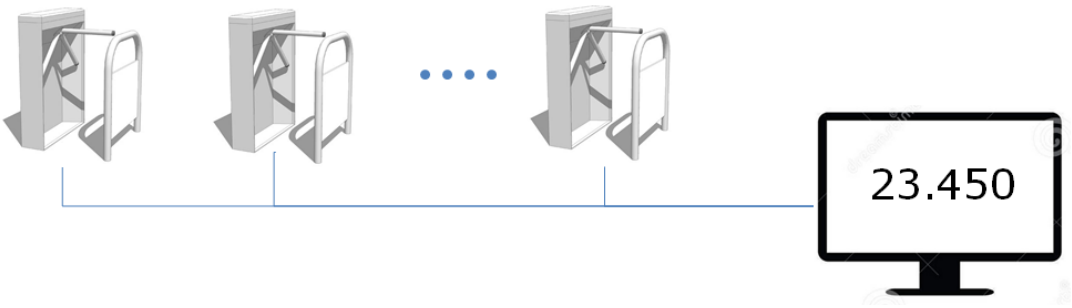
Exercise-3 (Blocking the GUI-thread)

Execute the main method in `ex3.BallDemo.java`
What is the obvious problem with this solution?

Solve the problem by rewriting the Ball class to extend thread, so that the balls can be started (as shown to the right) and stopped, and such that one can terminate the program by pressing the cross

Exercises Threads-1 Spring 2016

Exercise 4 (race condition)



Use the code in the package ex4 as start code for this exercise. This program simulates a large football stadium with many turnstiles that each updates a shared counter, for each spectator that passes a turnstile.

Initially, the code is set up to simulate 40 turnstiles, each (running in a separate thread) simulates that 1000 spectators pases.

Execute the code and observe the reslult of the shared counter.

If the result is not as expected, solve the problem:

1) Using the traditional synchronization
2) Using a AtomicInteger
3) Using a java.util.concurrent.locks.ReentrantLock

Exercise 5 (race condition)

Execute the main code in either `ex5.BankApp.java` or `ex5.BankAppExecutor.java`

The program includes a method `executeTransactions ()` which executes a series of transactions on a shared account object and prints true / false depending on the expected outcome.

Identify places in the code where it "goes wrong".

What should the closing balance be?

Experiment: Increase the number of calls of executeTransactions () or the number of threads and observe the effect.

Observe that there are two versions of the main() method, one that creates threads in a traditional way, and one using an ExecutorService (the recommended way).

Solve the problem in the Bank program:

1) Using the traditional synchronization
2) Using a java.util.concurrent.locks.ReentrantLock

Test

Exercises Threads-1 Spring 2016

Exercise 6 (Blocking the GUI thread)

The GUI below has a button and a label

The idea is; that, a press on the Countdown button should start a count down from 20 to 0 and update the label once per second, such that after the first second it prints "Back 9" etc. until "Back: 0"

```
private void jButton1_actionPerformed(ActionEvent e)
 {
   for (int i = 20; i >= 0; i--)
   {
     jLabel1.setText("Remains: " + i);
     try
     {
       Thread.sleep(1000);
     }
     catch (InterruptedException e)
     {
     }
   }
 }
```

Implement the program and test

What are the obvious problems (at least two) of the solution?

Solve the problem

Hints:
When you have solved the "blocking" problem, use a SwingWorker to get a Thread-safe solution when updating the GUI.

Ref: Use the example in the section "Updating the GUI from a Running Thread" from http://www.javacodegeeks.com/2012/12/multi-threading-in-java-swing-with-swingworker.html as a template.