Week 4. Lab exercise
The Threads Race

Felipe Orihuela-Espina

Contents

1 Problem statement

In this exercise you are going to program a thread race. In this race, there will be n participants and 1 referee. We shall simulate a race of 10.5 Km with 2 participants **but the program should support any number of participants and any length**, that is do NOT hardcode the partipants but instead rely on loops to the number of participants registered.

The participants are equal, e.g. of the same class, but they will simply have difference dorsal assigned by the referee upon registering to the race. They are simple objects implementing the `Runnable` interface. Once that they are allowed to start the race, they will decrement a counter of the remaining length to be run, and announce everytime they pass a kilometer mark until they cross the finish line tape.

The referee will be in charge of setting up the race for a given distance, register the participants, wrapping the participants into threads, starting the race and finally announcing the winner.

If there is any issue during the run (e.g. any exception) the race will be ended without declaring a winner.

2 What to submit?

This is a formative assignment. It is not assessed. You do NOT need to submit anything!

3 Synchronization

Once the race has started, i.e. *after* the referee exists the execution of the `Referee.go` method, the referee will await for the winner of the race to arrive. The detection of the winner will use two synchronization mechanisms;

- First, a method synchronization: `Referee.finished` method is `synchronized`, so that its execution cannot be interrupted.
- Second, the Waiting thread state. The condition to be waited is for the first participant to cross the finish line. For this we will rely on a `CountDownLatch`. Initialize your `CountDownLatch` to wait for 1 event (i.e. the winner), i.e. in the `Referee`'s constructor;

  ```
  finishLineTape = new CountDownLatch(1);
  ```

  ... and then in the `Referee.run` simply make the referee to `await` on the `CountDownLatch` variable;

  ```
  try {
      ...
      finishLineTape.await();
      ...
  } catch (InterruptedException e) {
      ...
      Thread.currentThread().interrupt();
  }
  ```

4 Class Partipant

Class `Participant` implements the `Runnable` interface. Complete the missing methods. Once the count down of the remaning length reaches 0, the Participant will tell the referee that it has `finished`.

```
public class Participant implements Runnable {
    private int dorsal = 0;
    private int raceLength = 0;//in [meters]
    private int remainingLength = 0;//in [meters]
    private Referee theReferee = null;

    public Participant (Referee raceReferee, int newDorsal, int newRaceLength)
    {
        //TO BE COMPLETED
    }

    //No getters and setters except for the getDorsal (intentional)
    public int getDorsal(){return dorsal;}

    //Implementation of interface Runnable
    public void run()
    {
        //TO BE COMPLETED
        //1. Announce that you are starting the race
        //2. Run the race; loop decrementing the remaining length 1 meter
        //      at a time and announce the crossing of each km mark.
        //3. Announce that you have finished the race
    }
}
```

5 Class Referee

Class `Referee` extends class `Thread` and contains the `main` method. Complete the missing methods.

```
import java.util.List;
import java.util.ArrayList;
import java.util.concurrent.CountDownLatch;

public class Referee extends Thread {

    private int raceLength = 0;
    private int nParticipants = 0;
    private List<Participant> participants = null;
    private List<Thread> participantsThreads = null;
```

```
private CountDownLatch finishLineTape = null;
private String winner =null;


public Referee(int newRaceLength)
{
    //TO BE COMPLETED
}


//Getters/Setters
public void setNumberOfParticipants(int nParticipants)
{ this.nParticipants = nParticipants;}


//Ready: Register participants
public void ready()
{
    //TO BE COMPLETED
    //Create the n participants and add them to the list of participants.
    System.out.println("Referee: Ready!");
}

//Set: Create the wrapping threads for the participants
public void set()
{
    //TO BE COMPLETED
    //Create the wrapping threads for each participant and
    //add them to the list of participants threads.
    System.out.println("Referee: Set!");
}

//Go: Start the wrapping threads of the participants
public void go()
{
    System.out.println("Referee: Go!");
    //TO BE COMPLETED
    //Start the participants.
}
```

```
    public synchronized void finished(int participantDorsal) {
        //TO BE COMPLETED
    }

    public void registerParticipant(int newDorsal)
    {
        //TO BE COMPLETED
    }

    public void run()
    {
        //TO BE COMPLETED
    }


    public static void main(String[] args)
    {
        Referee ref = new Referee(10500);
        try{
            ref.start();
            ref.join();
        }catch(InterruptedException e)
        { System.out.println("Race interrupted."); }
        System.out.println("Race ended.");
    }
}
```

6 Exemplary program output

The following is an exemplary valid output of the program.

```
Referee: Race length 10.5 km.
Referee: Registering 2 participants.
Referee: Registering participant with dorsal 1. 1 participants registered thus far.
Referee: Registering participant with dorsal 2. 2 participants registered thus far.
Referee: Ready!
Referee: 2 participants registered.
Referee: Participant 1 thread initiated.
Referee: Participant 2 thread initiated.
Referee: 2 participants set.
```

```
Referee: Set!
Referee: Go!
Referee: Launching participant 1.
Referee: Launching participant 2.
Participant 1: Starting race.
Participant 2: Starting race.
Participant 1: Passed km 1.
Participant 2: Passed km 1.
Participant 1: Passed km 2.
Participant 2: Passed km 2.
Participant 1: Passed km 3.
Participant 1: Passed km 4.
Participant 2: Passed km 3.
Participant 1: Passed km 5.
Participant 2: Passed km 4.
Participant 1: Passed km 6.
Participant 2: Passed km 5.
Participant 1: Passed km 7.
Participant 2: Passed km 6.
Participant 1: Passed km 8.
Participant 2: Passed km 7.
Participant 2: Passed km 8.
Participant 2: Passed km 9.
Participant 2: Passed km 10.
Participant 1: Passed km 9.
Participant 2: Finished race.
Participant 1: Passed km 10.
Referee: The winner is Participant 2
Race ended.
Participant 1: Finished race.

Process finished with exit code 0
```