Aufgabe 1

Beantworten Sie die Kontrollfragen A und B auf den Folien 18 und 32 des Inputs zu OOP10, falls diese noch nicht beantwortet wurden.

Siehe PRG2_OOP10_KF.docx / PRG2_OOP10_KF.pdf

Aufgabe 2

Man möchte den Wait-Pool eines Objektes testen mit folgenden Klassen.

```
public class MyThread extends Thread
{
  private Object lock;
  public MyThread(Object lock) {
    this.lock = lock;
  }
  @Override
  public void run () {
    System.out.println("warten...");
    synchronized (lock) {
      try {
        wait();
      } catch (InterruptedException e) {
        e.printStackTrace();
      }
    }
    System.out.println("...aufgewacht");
  }
}

public class TestWaitPool
{
  public static void main(String[] args) {
    Object lock = new Object();
    new MyThread(lock).start();
    try {
      Thread.sleep(1000);
      lock.notify();
    } catch (InterruptedException e) {
      e.printStackTrace();
    }
  }
}
```

Fragen:

- Was passiert bei der Ausführung von TestWaitPool ?

  Error!

  Exception in thread "Thread-0" java.lang.IllegalMonitorStateException

  warten...

  at java.lang.Object.wait(Native Method)

  at java.lang.Object.wait(Object.java:485)

  at uebungen.waitpool.MyThread.run(MyThread.java:17)

  Exception in thread "main" java.lang.IllegalMonitorStateException

  at java.lang.Object.notify(Native Method)

  at uebungen.waitpool.TestWaitPool.main(TestWaitPool.java:12)

  Java Result: 1

- Wie erklären Sie sich das Verhalten der Klassen?

  lock.notify() wird nicht innerhalb von einem synchronized block ausgeführt!

- Welche Korrekturen sind nötig?

  synchronized (lock) {

  lock.notify();

  }

Aufgabe 3

Ein praktischer Synchronisationsmechanismus ist das Latch. Latches sperren so lange, bis sie einmal ausgelöst werden und danach sind sie frei passierbar. Sie sollen nun ein Latch in Java schreiben und testen. Dazu soll ein Interface Synch verwendet, bzw. implementiert werden.

```
public interface Synch
{
  public void acquire() throws InterruptedException;
  public void release();
}
```

Für den Test veranstalten wir ein kleines Pferderennen. Dazu gibt es Pferde…

```
public class RaceHorse implements Runnable
{
  private Synch startSignal;
  private int nr;
  public RaceHorse(int nr, Synch startSignal)
  {
    this.nr = nr;
    this.startSignal = startSignal;
  }
  public void run()
  {
    try {
      startSignal.acquire();
      Thread.sleep((long)(3000.0*Math.random()));
    } catch (InterruptedException e) {
      e.printStackTrace();
    }
    System.out.println("Rennpferd "+nr+" ist im Ziel.");
  }
}
```

… und eine Rennbahn.

```
public class RaceHorse implements Runnable
public class Turf
{
  public static void main(String[] args)
  {
    Synch starterBox = new Latch();
    for (int i = 1; i < 6; i++)
      new Thread(new RaceHorse(i,starterbox)).start();
    starterBox.release();
  }
}
```

Damit alle Pferde gerecht gestartet werden, kommen diese in eine Starterbox. Sobald diese geöffnet wird, sollen die Pferde loslaufen.

```
public class Latch implements Synch
{
  private boolean latched = false;
  public void acquire() throws InterruptedException
  {
    //...Latch anfordern
  }
  public void release()
  {
    //...Latch freigeben
  }
}
```

Implementieren Sie anhand der Vorgaben die Klasse Latch.

```
    public synchronized void acquire() throws InterruptedException
    {
        while ( ! latched) {
            wait();
        }
    }

    public synchronized void release()
    {
        latched = true;
        notifyAll();
    }
```

Frage: Ist das Rennen wirklich gerecht?

Nein, es läuft ja nur „quasi parallel“, notifyAll() ist nicht zu 100% „synchron“ für alle Threads.