Aufgabe 1

Beantworten Sie die Kontrollfragen A bis C auf den Folien 11, 25 und 37 des Inputs zu OOP8, falls diese noch nicht beantwortet wurden.

Siehe PRG2_OOP8_KF.docx / PRG2_OOP8_KF.pdf

Aufgabe 2

Im folgenden Beispiel berechnet die Klasse Calculate eine Summe. Der Thread t führt die Addition nebenläufig durch, der Main-Thread gibt das Resultat aus.

```
public class Calculate implements Runnable
{
  private long sum = 0;
  private boolean fini;
  public void run()
  {
    for (int i = 0; i <= 100000000; i++) {
      sum += i;
    }
    fini = true;
  }
  public long sum()
  {
    return sum;
  }
  public boolean finished()
  {
    return fini;
  }
}

public class SimpleBlocking
{
  public static void main(String[] args)
  {
    Calculate calc = new Calculate();
    Thread t = new Thread(calc);
    t.start();

    // Version 1
    while (!calc.finished());
    try {
      // Version 2
      Thread.sleep(1);
      // Version 3
      t.join();
    } catch (InterruptedException ex) { }
    System.out.println("Summe = " + calc.sum());
  }
}
```

Starten Sie das Programm SimpleBlocking mit jeweils einer Version der einfachen Blockierung.

Fragen:

- Funktionieren alle drei Versionen?
  nein
- In welchen Zuständen befinden sich die Threads in den drei Versionen?
  new, runnable, blocked, waiting, timed_waiting, terminated
- Bewerten Sie die drei Versionen. Welche ist die beste Variante?
  Version 1 und 3 funktionieren, jedoch „loopt" der Prozess bei Version 1 unnötig
  ⇨ *Version 3 ist die beste Variante.*

Erstellen Sie einen neuen Thread s, der die Ausgabe der Summe übernimmt.

Frage: Was stellen Sie jetzt fest?

Wenn die Threads nicht synchronisiert (join) werden, stimmt die Ausgaben nicht.

Beenden eines Threads (für Interessierte)

Allgemein ist ein Thread beendet, wenn eine der folgenden Bedingungen zutrifft:

- Die run-Methode wurde ohne Fehler beendet.
- In der run-Methode tritt eine Ausnahme (Exception) auf, welche die Methode beendet.
- Der Thread wurde von aussen abgebrochen.

Die dritte Möglichkeit, den Thread von aussen abzubrechen, soll das folgende Beispiel zeigen. Dabei wird die ThreadAddition abgebrochen, wenn die Berechnung länger als eine Sekunde dauert. Dazu werden die Methoden interrupt und isInterrupted verwendet, mit deren Hilfe man ein Statusflag des Threads setzen und abfragen kann. Statt dass in der for-Schlaufe nur die Zahlen zur Summe addiert werden, überprüft die if-Selektion mit isInterrupted den Zustand des Statusflags. Gibt die Methode true zurück, wurde das Statusflag gesetzt und die run- Methode sollte beendet werden.

```
public class ThreadAddition extends Thread
{
  private int n;
  private String id;
  public ThreadAddition(String id, int n)
  {
    this.id = id;
    this.n = n;
  }
  @Override
  public void run()
  {
    long sum = 0;
    for (int i = 0; i <= n; i++) {
      if (isInterrupted()) {
        break;
      }
      else {
        sum += i;
      }
    }
    if (!isInterrupted()) {
      System.out.println(id +"-"+getName()+": SUM"+n+" -> "+sum);
    }
    else {
      System.out.println(id + "-" + getName() + ": interrupted.");
    }
  }
}

public class TestThreadAddition
{
  public static void main(String[] args)
  {
    ThreadAddition work1 = new ThreadAddition("A", 100000000);
    ThreadAddition work2 = new ThreadAddition("B", 100000);
    ThreadAddition work3 = new ThreadAddition("C", 100);
    work1.start();
    work2.start();
    work3.start();
    try {
      Thread.sleep(1000);
      work1.interrupt();
      work2.interrupt();
      work3.interrupt();
    }
    catch (InterruptedException e) {
      System.out.println(e);
    }
  }
}
```

Wie das Ergebnis zeigt, wurden die Threads mit kleinen Zahlen nicht unterbrochen, da sie innerhalb einer Sekunde bereit beendet waren, im Gegensatz zu Thread A, der nach einer Sekunde mit der Berechnung noch nicht zu Ende war.

```
B-Thread-1: SUM100000 -> 5000050000
C-Thread-2: SUM100 -> 5050
A-Thread-0: interrupted.
```

Aufgabe 3 (freiwillig, aber sehr interessant)

Die Klasse `ThreadAddition` soll bei der Berechnung etwas gebremst werden. Nach jeder Addition soll der Thread 15msec warten, bis die nächste Addition durchgeführt wird. `ThreadAddition` soll mit Hilfe von `interrupt` und `isInterrupted` vorzeitig beendet werden können.

```
public class ThreadAddition extends Thread
{
  private int n;
  private String id;
  public ThreadAddition(String id, int n)
  {
    this.id = id;
    this.n = n;
  }
  @Override
  public void run()
  {
    long sum = 0;
    for (int i = 0; i <= n; i++) {
      try {
        if (isInterrupted()) {
          break;
        }
        else {
          sum += i;
        }
        Thread.sleep(13);
      } catch (InterruptedException e) { }
    }
    if (!isInterrupted()) {
      System.out.println(id+"-"+getName()+": SUM"+n+" -> "+sum);
    }
    else {
      System.out.println(id + "-" + getName() + ": interrupted.");
    }
  }
}
```

Für den Test nehmen Sie die bereits gezeigt Klasse TestThreadAddition.

Fragen:

- Was stellen Sie beim Ausführen der drei Threads, und vor allem beim Beenden fest ?

Einzig Thread C beendet sich, die anderen zwei bleiben „hängen“.

- Wie erklären Sie sich das Programmverhalten?

Es wird eine Exception ausgelöst, diese wird jedoch nicht behandelt. (d.h. abgefangen aber nichts unternommen)

- Wie müsste eine korrekte Lösung aussehen?

Bei try-catch müsste bei einer Interrupt Exception auch wirklich abgebrochen werden – andernfalls wird der Interrupt einfach „ignoriert“.

```
catch (InterruptedException e) {
  break;
}
```