Aufgabe 9 [12 Punkte] **Threads**

a) [4] Beantworten Sie die folgenden Teilaufgaben.

Erläutern Sie die Zustände `ready`, `running`, `sleeping`, `dead`, in denen sich ein Thread befinden kann.

Was ist ein Deadlock?

b) [8] Es soll die Summe eines Arrays von ganzen Zahlen mithilfe mehrerer Threads berechnet werden. Schreiben Sie dazu eine Methode `public static sum(int[] input, int numThreads)`, die die Summe der Einträge des Arrays `input` zurückliefert.

Das Array soll dabei in `numThreads` möglichst gleich große disjunkte Abschnitte aufgeteilt werden und für jeden Abschnitt soll die Summe der Einträge in diesem Abschnitt in einem extra Thread berechnet werden. Die Summe aller Einträge des Arrays `input` ergibt sich dann entsprechend aus der Summe über die Ergebnisse der Abschnitte.

Kann das Array nicht in `numThreads` exakt gleich große Abschnitte aufgeteilt werden, sollen alle bis auf den letzten Abschnitt $\lfloor$`input.length`$/$`numThreads`$\rfloor$ groß sein. Beispiel: Die Aufteilung für `input.length`$= 5$ und `numThreads`$= 3$ ist $1, 1, 3$, da $\lfloor 5/3 \rfloor = 1$. Sie können annehmen, dass `numThreads` kleiner als `input.length` ist.

Aufgabe 7 [9 Punkte] **Threads**

Gegeben sei der folgende Code:

```
1  public class Konflikt extends Thread {
2
3      static Integer number = 1;
4      int id;
5      int tmp;
6
7      public Konflikt(int id) {
8          this.id = id;
9          this.start();
10     }
11
12     public static void main(String[] args) {
13         for (int i = 1; i <= 3; i++) {
14             new Konflikt(i);
15         }
16     }
17
18     public void run() {
19         read();
20         compute();
21         write();
22     }
23
24     public void read() {
25         synchronized (number) {
26             tmp = number;
27         }
28     }
29
30     public void write() {
31         synchronized (number) {
32             number = tmp;
33         }
34     }
35
36     public void compute() {
37         System.out.println("" + tmp);
38         if (id == 1) {
39             tmp = tmp + 2;
40         } else if (id == 2) {
41             tmp = tmp * 3;
42         } else {
43             tmp = tmp * 2;
44         }
45     }
46
47 }
```

Geben Sie alle möglichen Ausgaben an, die durch Zeile 37 vom jeweiligen Thread mit der gegebenen `id` erzeugt werden können:

Konflikt.id==1: _____

Konflikt.id==2: _____

Konflikt.id==3: _____

Aufgabe 8 [13 Punkte] **Threads**

In dieser Aufgabe soll ein *paralleles Summieren auf einem Baum* implementiert werden, d.h. die Summe aller `data`-Attribute in einer durch `Tree` gegebenen Datenstruktur soll berechnet werden:

```
public class Tree
{
   public int data;
   public Tree[] children; //always !=null
}
```

Im Folgenden sollen Sie im Codetemplate auf der nächsten Seite die Klasse `SumThread` vervollständigen:

- Implementieren Sie die Methode `run`. Sie soll die Summe aller `data`-Attribute im `Tree` `t` berechnen. Dabei sollen die Summen der `data`-Attribute der Kinder von `t` parallel berechnet werden. Das Ergebnis soll in `result` gespeichert werden.
- Sie können davon ausgehen, dass `this.children!=null` in allen `Tree`s gilt.

Hinweis: Bei korrekter Implementierung lautet die Ausgabe des Programms am Ende des Codetemplates „`Sum = 15`“, da $5 + 0 + 1 + 2 + 3 + 4$ berechnet wird.

```
class SumThread extends Thread{
   int result;
   Tree t;
   SumThread(Tree t){
      this.t=t;
   }

   public static void main(String args[]) {
      Tree t = new Tree();
      t.data = 5;
      t.children = new Tree[5];
      for(int i = 0; i < 5; i++){
         t.children[i] = new Tree();
         t.children[i].data = i;
         t.children[i].children = new Tree[0];
      }
      SumThread st = new SumThread(t);
      st.start();
      try {
         st.join();}
      catch(InterruptedException ie) {}
      System.out.println("Sum = "+st.result);
   }
}
```

Data race

Betrachten Sie die folgenden beiden Klassen Incrementer und IncrementerTest:

```
8   public class Incrementer extends Thread {
9       private int[] data;
10      public Incrementer(int[] data) {
11          this.data = data;
12      }
13      @Override
14      public void run() {
15          int temp = data[0];
16          System.out.println(this.getName()+": getesteter Wert "+temp);
17          while (temp < 2) {
18              temp = data[0];
19              System.out.println(this.getName()+": gelesener Wert "+temp);
20              data[0] = temp+1;
21              System.out.println(this.getName()+": geschriebener Wert "+(temp+1));
22              temp = data[0];
23              System.out.println(this.getName()+": getesteter Wert "+temp);
24          }
25      }
26  }
```

```
8   public class IncrementerTest {
9       public static void main(String[] args) {
10          int[] data = {0};
11          Incrementer incA = new Incrementer(data);
12          Incrementer incB = new Incrementer(data);
13          incA.start();
14          incB.start();
15          try {
16              incA.join();
17              incB.join();
18          } catch (InterruptedException e) {} finally {
19              System.out.println("data[0]: "+data[0]);
20          }
21      }
22  }
```

Eine mögliche Ausgabe von IncrementerTest ist

```
Thread-0: getesteter Wert 0
Thread-1: getesteter Wert 0
Thread-0: gelesener Wert 0
Thread-1: gelesener Wert 0
Thread-0: geschriebener Wert 1
Thread-0: getesteter Wert 1
Thread-1: geschriebener Wert 1
Thread-0: gelesener Wert 1
Thread-1: getesteter Wert 1
Thread-1: gelesener Wert 1
Thread-0: geschriebener Wert 2
Thread-1: geschriebener Wert 2
Thread-1: getesteter Wert 2
Thread-0: getesteter Wert 2
data[0]: 2
```

Geben Sie zwei weitere mögliche Ausgaben von IncremeterTest an, deren letzte Zeile jeweils einmal "data[0]: 2" und einmal "data[0]: 3" ist.