Bearbeitungszeitraum: eine Woche

a) Implementieren Sie eine Lese – Schreibsperre.
Benutzen Sie dazu die Klasse ReadWriteLock aus den java.util.concurrent Paketen.
Implementieren Sie eine Speicher Klasse, auf die von verschiedenen Leser und Schreiber Objekten aus gleichzeitig zugegriffen werden soll. Die Klasse Speicher ist eine Liste, in der einzelne Zeilen als Strings gespeichert werden:

```
class Speicher extends LinkedList<String> {
      void writeLine(String zeile)
      // in writeLine soll der Liste eine Zeile hinzugefügt
      // werden
      String readLine(int zeilenNummer)
      // in readLine soll der Listeneintrag mit dem Index
      // zeilenNummer gelesen und zurückgegeben werden
      synchronized void schreiberRein()
      // soll die Zahl schreibendeThreads in Speicher um
      // eins erhöhen, und die aktuellen Zahlen
      // lesendeThreads und schreibendeThreads
      // auf Bildschirm
      // ausgeben. Soll immer aufgerufen werden, wenn ein
      // Thread die Methode writeLine betritt.
      synchronized void schreiberRaus()
      // soll die Zahl schreibendeThreads in Speicher um
      // eins erniedrigen, und die aktuellen Zahlen
      // lesendeThreads und schreibendeThreads
      // auf Bildschirm
      // ausgeben. Soll immer aufgerufen werden, wenn ein
      // Thread die Methode writeLine verläßt
      synchronized void leserRein()
      // analog zu schreiberRein
      synchronized void leserRaus()
      // analog zu schreiberRaus
}
```

Die Klasse Leser soll den Inhalt von Speicher zeilenweise lesen und auf dem Bildschirm ausgeben:

```
class Leser extends Thread {
   Leser(Speicher speicher)
   public void run()
```

Die Klasse Schreiber soll in Speicher zeilenweise Strings schreiben.

```
class Schreiber extends Thread {
   Schreiber(Speicher speicher, String[] text, int
zahlWiederholungen)
   public void run()
}
```

I) Implementieren Sie eine Main Klasse, in der zwei Schreiber Objekte gleichzeitig jeweils text1 und text2 in ein Speicher Objekt schreiben:
```
String[] text1 = {"text 1: Zeile 1", "text 1: Zeile 2"};
String[] text2 = {"text 2: Zeile 1", "text 2: Zeile 2"};
```
Dabei sollen die Schreiber die Texte zweimal herausschreiben. Gleichzeitig sollen zwei Leser Objekte den Inhalt lesen. Demonstrieren Sie anhand der Ausgaben von schreiberRaus, schreiberRein, leserRaus, leserRein dass ohne Locks gleichzeitig mehr als ein Schreiber oder auch ein Schreiber und ein Leser sich im Speicher Objekt aufhalten können.
Wo kann man in `readLine()` und in `println()` geschickt ein `Thread.yield()` einsetzen, um ein möglichst häufiges Umschalten zu erreichen?

II) Führen Sie bei den Schreibern und Lesern ein `ReentrantReadWriteLock` Objekt ein. Demonstrieren Sie, dass damit maximal
ein Schreiber aber auch

III) mehrere Leser gleichzeitig den Speicher betreten können.

IV) Erhöhen Sie die Anzahl der Schreibwiederholungen bei den Schreibern auf eine große Zahl (z.B. 2000) und kommentieren Sie die lock Befehle und auch die Aufrufe von schreiberRaus, schreiberRein, leserRaus, leserRein aus.

- Beobachten Sie, dass Exceptions geworfen werden? Wenn ja, welche? Zeigt Ihnen Ihre Entwicklungsumgebung an, in welchem Thread (Leser oder Schreiber) der Fehler auftritt? Bei welchem Index der Liste?
- Woran liegt der Fehler? Lesen Sie auch in der API zu LinkedList besonders den Abschnitt, der durch „Note that this implementation is not synchronized.“ eingeleitet wird.

V) Kommentieren Sie die Aufrufe von schreiberRaus, schreiberRein, leserRaus, leserRein wieder ein. Treten die Exceptions immer noch auf? Wenn nicht: woran könnte das liegen?

b) Implementieren Sie das Philosophen Problem:

I) ohne Semaphorengruppe. Schaffen Sie es, die Verklemmung beim 5 Philosophen Problem in einer Implementierung hervorzurufen (wenigstens manchmal)? Ihre Lösung darf nicht verklemmungsfrei sein.
Wenn ihre Lösung verklemmt: drücken Sie die Pause Taste ihrer Entwicklungsumgebung. Jetzt können Sie erkennen, wo die einzelnen Threads stehen geblieben sind. Ist nun erkennbar, wie die Verklemmung zustande gekommen ist?

II) mit Semaphorengruppe und verklemmungsfrei.
Sie können folgende Implementierung einer Semaphorengruppe unten in diesem Dokument herauskopieren und benutzen:

```
public class SemaphoreGroup {
   private int[] values;
   public SemaphoreGroup(int numberOfMembers) {
      if(numberOfMembers <= 0) {
         return;
      }
      values = new int[numberOfMembers];
   }

   public int getNumberOfMembers() {
      return values.length;
   }


   public synchronized void changeValues(int[] deltas) {
      if(deltas.length != values.length) {
         return;
      }
      while(!canChange(deltas)) {
         try {
            wait();
         } catch(InterruptedException e) {
            System.out.println("InterruptedException
aufgetreten");
         }
      }
      doChange(deltas);
      notifyAll();
   }

   private Boolean canChange(int[] deltas) {
      for(int i=0; i < values.length; i++) {
         if(values[i]+deltas[i] < 0) {
            return false;
         }
      }
      return true;
   }

   private void doChange(int[] deltas) {
      for(int i=0; i < values.length; i++) {
         values[i]=values[i]+deltas[i];
      }
   }
}
```