Bearbeitungszeitraum: zwei Wochen

a) Betrachten Sie das folgende Programm, das die Implementierung einer Speicherzelle für einen int-Wert darstellt:

```
class Speicherzelle {
   private int wert;
   public synchronized void setWert(int w ) {
       wert = w;
   }
   public synchronized int getWert() {
      return wert;
   }
   public synchronized void swapWert(Speicherzelle s) {
      int h = s.getWert();
      s.setWert(wert);
      setWert(h);
   }
}
```

Die Klasse darf geändert werden.
Führen mehrere Threads den obigen Programmcode gleichzeitig aus, so kann es zu einer Verklemmung kommen.
Programmieren Sie eine oder mehrere Thread Klassen, die eine solche Verklemmung hervorrufen - bei der Aufgabenabgabe soll die Verklemmung vorgeführt werden. Von den Methoden der Klasse Speicherzelle soll außerhalb der Klasse Speicherzelle nur die swapWert-Methode aufgerufen werden (nicht setWert oder getWert)!
Tip: um die Verklemmung hervorzurufen hilft es, die Methode swapWert etwas auszubremsen...

b) Gegeben sei die folgende Schnittstelle für EventDispatcher-Objekte:

```
public interface EventDispatcher {
   public void addActionListener
                  (java.awt.event.ActionListener al);
   public void processActionEvent
                  (java.awt.event.ActionEvent ae);
}
```

- Bei einem EventDispatcher-Objekt können sich ActionListener-Objekte mit der addActionListener-Methode beliebig oft registrieren.
- In der addActionListener Methode wird der übergebene ActionListener in einem ActionListener Feld Attribut gespeichert.
- Inhalt der processActionEvent-Methode: in einer Schleife wird das ActionListener Feld Attribut durchlaufen und die actionPerformed-Methode eines jeden dort gespeicherten ActionListener aufgerufen.
- Benutzen Sie das ActionListener interface aus dem Paket java.awt.event – ebenso die Klasse ActionEvent aus demselben Paket.

Aufgaben:

I) Implementieren Sie die Klasse EventDispatcherImpl als Unterklasse von EventDispatcher so, dass die Implementierung die oben beschriebenen Eigenschaften erfüllt!

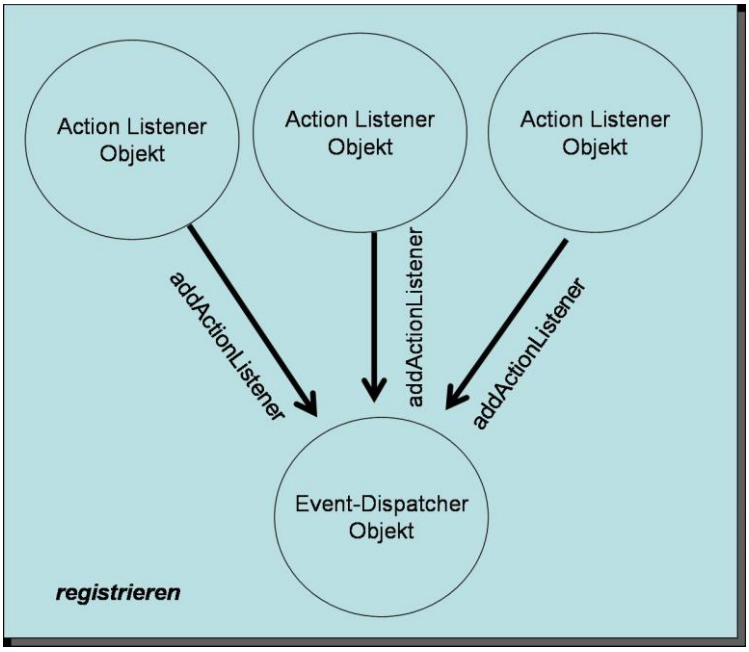
II) Implementieren Sie zwei Unterklassen von ActionListener:
   i) ActionListenerImpl1: Sie müssen die actionPerformed Methode implementieren. Diese Methode soll nur „ActionListener 1 wurde aufgerufen“ auf dem Bildschirm ausgeben.
   ii) ActionListenerImpl2: die actionPerformed Methode soll nur „ActionListener 2 wurde aufgerufen“ auf dem Bildschirm ausgeben.

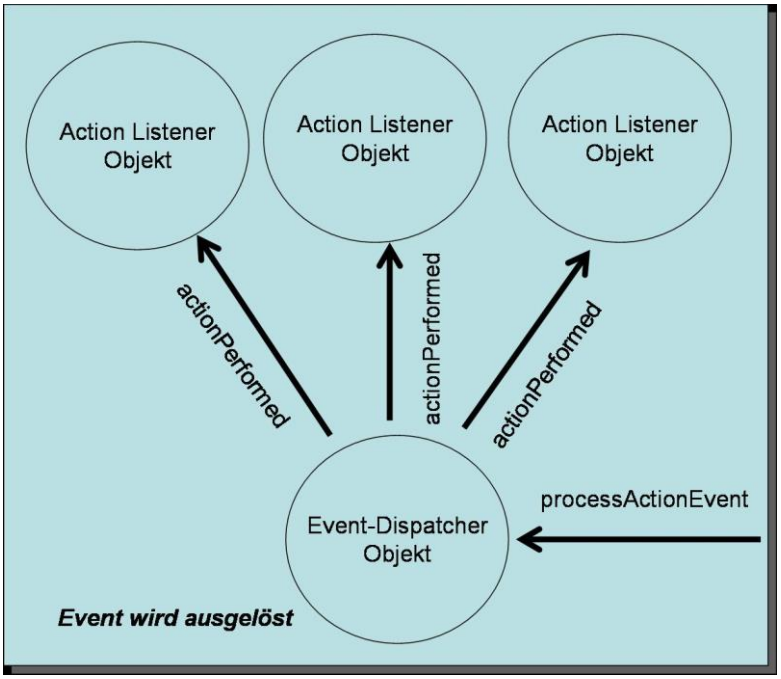
III) Implementieren Sie die main Methode:
   i) Erzeugen Sie ein Objekt vom Typ ActionListenerImpl1.
   ii) Erzeugen Sie ein Objekt vom Typ ActionListenerImpl2.
   iii) Erzeugen Sie ein Objekt vom Typ EventDispatcherImpl.
   iv) Registrieren Sie die beiden ActionListener Objekte beim EventDispatcher Objekt.
   v) Rufen Sie die processActionEvent Methode von EventDispatcherImpl auf. Statt eines ActionEvent Objekts übergeben Sie einfach null, da dieses Objekt nicht benutzt wird. Auf dem Bildschirm sollte jetzt folgendes ausgegeben werden:
   ActionListener 1 wurde aufgerufen
   ActionListener 2 wurde aufgerufen

IV) Erweitern Sie die Implementierung so, dass die actionPerformed-Methode jedes registrierten ActionListener-Objektes von einem eigenen Thread (Arbeitsthread) ausgeführt wird! Auf diese Weise sollen die actionPerformed-Methoden für die registrierten Objekte parallel ausgeführt werden. Sie müssen dafür nur die processActionEvent Methode ändern.

V) Beschreiben Sie für die parallele Programmversion von IV. zwei Situationen, in denen es passieren kann, dass während der Programmausführung die actionPerformed-Methode eines ActionListener-Objektes von zwei Arbeitsthreads eines EventDispatcher-Objektes gleichzeitig ausgeführt wird!

VI) Erweitern Sie die Implementierung so, dass die actionPerformed-Methode eines registrierten ActionListeners nicht von zwei verschiedenen Threads gleichzeitig ausgeführt werden kann!


Action Listener Objekt
Action Listener Objekt
Action Listener Objekt
addActionListener
addActionListener
addActionListener
Event-Dispatcher Objekt
registrieren



c) Wie viele Threads können bei Ausführung der main-Methode des folgenden Programms entstehen und gleichzeitig existieren?

```
class T extends Thread {
   T() {
      start();
   }
   public void run() {
      T.main(null);
   }
   public static void main(String[] argv) {
      new T();
   }
}
```

d) Gegeben sei das folgende Java-Programm:

```
public class A {
   private int wert;
   public A ( int startWert ) { wert = startWert; }
0   public void tausche (A a) {
1      int h = wert;
2      wert = a.wert;
3      a.wert = h;
   }
}
```

```
public class WertTausch extends Thread {
   private A b0;
   private A b1;
   WertTausch(A b0, A b1) {
      this.b0 = b0;
      this.b1 = b1;
   }
   public void run() {
      b0.tausche(b1);
   }
}
public class Main {
   private static A a0;
   private static A a1;
   public static void main(String[] args) {
      a0 = new A(0);
      a1 = new A(1);
      new WertTausch(a0, a1).start(); // Thread X
      new WertTausch(a1, a0).start(); // Thread Y
   }
}
```

Bearbeiten Sie bitte die folgenden Aufgaben:
In den Wert-Attributen der beiden A-Objekte a0 und a1 können
0 und 1 oder
1 und 1 oder
1 und 0 oder
0 und 0 stehen, wenn beide Threads ihre run-Methode beendet haben.
Finden Sie für alle diese vier Fälle einen Programmablauf (mit Threadumschaltungen) heraus, der das entsprechende Ergebnis produziert. Sie müssen **kein** Programm erstellen, das ihre Antwort demonstriert.
Erstellen Sie vier Versionen der folgenden Tabelle mit unterschiedlichen Abläufen und zwar jeweils einmal für das Ergebnis

i) 0 und 1
ii) 1 und 1
iii) 1 und 0
iv) 0 und 0

(das Programm bleibt dabei immer dasselbe!)

Tabellen**beispiel** für Ergebnis 0 und 1:
Die Tabelle zeigt die Werte der Variablen `wert` und `h` in `a0` und `a1` jeweils **nachdem** der Thread X oder Y die entsprechende Zeile ausgeführt hat. In diesem Beispiel findet nur eine Threadumschaltung statt, und zwar nachdem Thread X die 3. Zeile der Methode `tausche()` von Objekt `a0` durchgeführt hat.

Zeilennr. Thread		Objekt `a0`		Objekt `a1`	
X in `a0`	Y in `a1`	`wert`	`h`	`wert`	`h`
1	-	**0**	0	**1**	-
2	-	1	0	1	-
3	-	1	0	0	-
-	1	1	0	0	0
-	2	1	0	1	0
-	3	**0**	0	**1**	0

II) Betrachten Sie jetzt die folgende Variante der Klasse A

```
public class A {
   private int wert;
   private Object o = new Object();
   public A ( int startWert ) { wert = startWert; }
   public void tausche (A a) {
      synchronized (o) {
         int h = wert;
         wert = a.wert;
         a.wert = h;
      }
   }
}
```

Ändert sich das Verhalten des Programms gegenüber I)? Begründen Sie Ihre Antwort. Sie brauchen Ihre Antwort nicht anhand eines Programms zu demonstrieren.

III) Ändert sich das Verhalten des Programms gegenüber II), wenn die Deklaration desAttributs o geändert wird zu

```
private static Object o = new Object();
```

Begründen Sie auch jetzt Ihre Antwort. Demonstrieren Sie Ihre Antwort anhand eines Programms (siehe auch V).

IV) Sehen Sie sich eine weitere Variante der Klasse A an:

```
public class A {
   private int wert;
   public A ( int startWert ) { wert = startWert; }
   public synchronized void tausche (A a) {
      synchronized (a) {
         int h = wert;
         wert = a.wert;
         a.wert = h;
      }
   }
}
```

Welche der beiden Synchronisationsmethoden III) oder IV) halten Sie für geeigneter? Kann es bei einem oder beiden der genannten Verfahren zu Problemen kommen? Begründen Sie jeweils Ihre Antwort. Auch dafür muss kein Programm gemacht werden.

V) Ergänzen Sie die Methode main um Ausgabeanweisungen, die die Wert-Attribute der A-Objekte a0 und a1 am Bildschirm anzeigen und zwar einmal vor dem Starten der beiden Threads und nachdem beide Threads ihre run-Methode beendet haben. Ggf. müssen Sie dazu auch noch andere Klassen modifizieren. Vorhandene Modifikatoren (z.B. private) dürfen geändert werden. Verwenden Sie die Variante der Klasse A aus Teil III).