— Übungsblatt 4 (T) —

Aufgabe 12 (T) (Swing-Komponenten und Threads)

a) Schreiben Sie eine Subklasse `BlinkButton` der Klasse `JButton`. Die Klasse `BlinkButton` soll (in Form eines Threads) die Beschriftung des Buttons „blinkend“ darstellen, indem der Button-Text und ein vorgegebener alternativer Text ständig vertauscht werden.

Gehen Sie beim Aufbau der Klasse wie folgt vor:

- Vereinbaren Sie eine Klassenvariable `tauschText`, die zunächst den alternativen Button-Text `"Hoppala"` enthält.
- Deklarieren Sie einen Konstruktor mit einem `String`-Parameter, der den entsprechenden Konstruktor der Klasse `JButton` aufruft und anschließend einen mit dem eigenen Objekt verbundenen Thread erzeugt und startet.
- Schreiben Sie eine `run`-Methode, in der innerhalb einer geeigneten Schleife jeweils
  - der Blink-Thread eine Sekunde lang schläft (eine `InterruptedException` soll dabei abgefangen werden) und danach
  - die aktuelle Button-Beschriftung und der Tausch-Text vertauscht werden.

b) Schreiben Sie eine Klasse für einen Test-Frame, der lediglich einen `BlinkButton` mit der Beschriftung `"Klausur"` aufweist.

Aufgabe 13 (T) (Thread-Synchronisation)

Nachfolgende Klassen simulieren zwei Terminals in Vorverkaufsstellen für Konzertkarten, an denen Karten gekauft werden können.

Mit dieser Realisierung ist beabsichtigt, dass die `main`-Methode eine Ausgabe der Form

```
Karten-Terminal 2: Sitzplatz 1 verkauft
Karten-Terminal 2: Sitzplatz 2 verkauft
Karten-Terminal 1: Sitzplatz 3 verkauft
Karten-Terminal 2: Sitzplatz 4 verkauft
...
```

erzeugt, d. h. dass die Sitzplätze der Reihe nach verkauft eine Karte für einen bestimmten Sitzplatz aber nur genau einmal verkauft wird, unabhängig vom Verkaufs-Terminal, an dem sie gekauft wird.

Die Sitzplatznummern der freien Plätze beziehen die Terminals von einem Objekt des Typs `KonzertDaten`. Dieses Objekt soll einen sehr einfachen Datenbankserver (und die entsprechneden Zugriffe auf eine Konzertkarten-Datenbank) simulieren.

Die Karten werden durch die sukzessiven Aufrufe der Methode `freierPlatz` der Reihe nach verkauft.

```
class KonzertDaten {
  private int sitzPlatz = 0;

  int freierPlatz() {
    int n = sitzPlatz;
    long t = System.currentTimeMillis() + 50, s = t + 50;
    while (System.currentTimeMillis() < t)
      /* simuliere einen Datenbankzugriff */   ;
    Thread.yield();  // erlaube Aktivitäten anderer Threads
    while (System.currentTimeMillis() < s)
      /* simuliere einen Datenbankzugriff */   ;
    return sitzPlatz = n + 1;
  }
}

class KartenTerminal extends Thread {
  private KonzertDaten daten;

  KartenTerminal(String name, KonzertDaten daten) {
    super(name);
    this.daten = daten;
  }
  public void run() {
    for (int i = 0; i < 100; i++)
      System.out.println(getName() + ": Sitzplatz " +
                         daten.freierPlatz() + " verkauft");
  }
}

public class UseTerminals {
  public static void main(String[] args) {
    KonzertDaten daten = new KonzertDaten();
    KartenTerminal t1 = new KartenTerminal("Karten-Terminal 1", daten),
                   t2 = new KartenTerminal("Karten-Terminal 2", daten);
    t1.start();
    t2.start();
  }
}
```

a) Liefert das Programm tatsächlich immer dieses Ausgabe, oder könnten sich auch andere Ausgaben ergeben?

b) Versuchen Sie die simultane Abarbeitung der Anweisungen der beiden Threads `t1` und `t2` zu beschreiben.

c) Wie kann man das unvorhersehbare Verhalten des Programmes beseitigen?

Aufgabe 14 (T) (Thread-Synchronisation und -Kommunikation)

Gegeben seien die nachfolgenden Klassen.

```
class Wert {
  private int wert;

  public int get(int verbraucherNr) {
    System.out.println("Verbraucher Nr. " + verbraucherNr + " got: " + wert);
    return wert;
  }

  public void put(int wert, int erzeugerNr) {
    System.out.println("Erzeuger    Nr. " + erzeugerNr + " put: " + wert);
    this.wert = wert;
  }
}
```

```
class Erzeuger extends Thread {
  private Wert w;

  private int nr;

  public Erzeuger(Wert c, int n) {
    w  = c;
    nr = n;
  }

  public void run() {
    for (int i = 0; i < 10; i++) {
      w.put(i, nr);
      try {
        sleep((int)(Math.random() * 100));
      }
      catch (InterruptedException e) {
      }
    }
  }
}
```

```
class Verbraucher extends Thread {
  private Wert w;
  private int nr;

  public Verbraucher(Wert c, int n) {
    w = c;
    nr = n;
  }

  public void run() {
    for (int i = 0; i < 10; i++) {
      int v = w.get(nr);
      try {
        sleep((int)(Math.random() * 100));
      }
      catch (InterruptedException e) {
      }
    }
  }
}
```

```
public class EVTest {
  public static void main (String args[]) {
    Wert         c = new Wert();
    Erzeuger    e1 = new Erzeuger(c, 1);
    Erzeuger    e2 = new Erzeuger(c, 2);
    Verbraucher v1 = new Verbraucher(c, 1);
    Verbraucher v2 = new Verbraucher(c, 2);
    e1.start();
    e2.start();
    v1.start();
    v2.start();
  }
}
```

---

a) Erwünscht ist, dass von jedem Erzeuger genau die Zahlen 0 bis 9 erzeugt werden und diese dann auch alle genau einmal von den Verbrauchern verbraucht werden. Bestimmen Sie zwei mögliche Fehler im Programm.

b) Die Methoden `get` und `put` seien nun mit dem Modifikator `synchronized` deklariert. Wie verändert sich das Verhalten des oben angegebenen Programmes?

c) Führen Sie eine logische Instanz-Variable `verfuegbar` in der Klasse `Wert` ein. Diese soll den Wert `true` haben, sobald ein Erzeuger eine Zahl erzeugt hat, und der Wert soll unmittelbar nach dessen Verbrauch `false` sein. Gewährleisten Sie, dass jetzt wirklich jede der Zahlen 0 bis 9 von jedem Erzeuger genau einmal erzeugt und gleich im Anschluss verbraucht wird.

Aufgabe 15 (T)

(Threads nachvollziehen)

Mit den nachfolgenden Klassen ist beabsichtigt, dass beim Start der `main`-Methode die Zahlen 1 bis 4 und die zugehörigen Zahlenpaare 11, 22, 33 und 44 paarweise untereinander auf das Konsolenfenster ausgegeben werden. Zwei der Paare sollen vom `WN`-Thread A, die anderen vom `WN`-Thread B ausgegeben werden. Leider arbeiten die beiden Threads aber bezüglich der Methode `next` nicht synchronisiert, und sie machen zwischen den Ausgaben ab und zu Pause.

Vollziehen Sie den parallelen Programmablauf nach und tragen Sie unter Berücksichtigung der Pausen die Bildschirm-Ausgaben in ihrer zeitlichen Reihenfolge in die Tabelle ein.

Achtung: In der Tabelle (eingeteilt im Sekundentakt) können sowohl leere Zellen, als auch Zellen mit einer oder mit zwei Ausgabezeilen vorkommen!

```
class Werte {
  private static int x = 0;
  public static void next() {
    x++;
    System.out.println(Thread.currentThread().getName() + ": x = " + x);
    try {
      Thread.sleep(2000);
    } catch (InterruptedException ie) { }
    System.out.println(Thread.currentThread().getName() + ": x+x = " + x+x);
  }
}
class WN extends Thread {
  int zeit;
  WN(String name, int zeit) {
    super(name);
    this.zeit = zeit;
  }
  public void run() {
    for (int i = 0; i < 2; i++)
      try {
        Thread.sleep(zeit);
        Werte.next();
      } catch (InterruptedException ie) { }
  }
}
public class WerteTest {
  public static void main(String[] args) {
    WN t1 = new WN("A", 1010);
    WN t2 = new WN("B", 2005);
    t1.start();
    t2.start();
  }
}
```

Zeitpunkt	Ausgabe
nach 1 sec:	
nach 2 sec:	
nach 3 sec:	
nach 4 sec:	
nach 5 sec:	
nach 6 sec:	
nach 7 sec:	
nach 8 sec:	