Praktikum Concurrency 1

1. Dreiecksfläche: Warmup mit Exceptions

Die Fläche eines Dreiecks mit den Seitenlängen `a, b, c` kann aus folgender Formel berechnet werden

```
flaeche = Math.sqrt(s * (s - a) * (s - b) * (s - c));
```

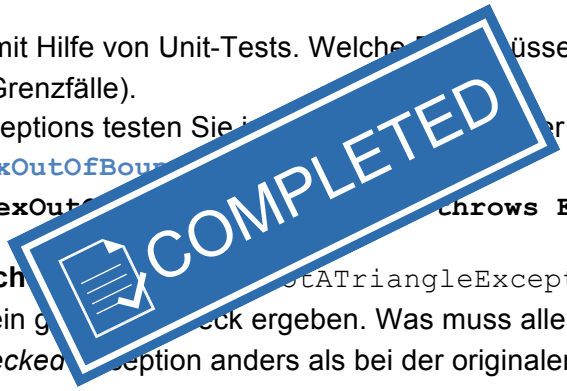
wobei `s = (a + b + c)/2` ist.

Aufgaben:

a) Schreiben Sie eine Methode, welche die Fläche eines Dreiecks aus den Seiten a, b, c (verwenden Sie den Typ `double`) berechnet. Die Methode soll eine **`IllegalArgumentException`** werfen, falls ungültige Werte übergeben werden oder die drei Seiten kein Dreieck darstellen.

b) Testen Sie Ihre Methode mit Hilfe von Unit-Tests. Welche [illegible] müssen getestet werden? (Vergessen Sie nicht die Grenzfälle).
Tipp: Das Werfen von Exceptions testen Sie [illegible] Annotation:

```
@Test(expected=IndexOutOfBou[illegible]
public void testIndexOut[illegible]hrows Exception { ... }
```



c) Erstellen Sie eine eigene **ch**[illegible]`tATriangleException`, die geworfen wird wenn die Angaben kein g[illegible]eck ergeben. Was muss alles angepasst werden? Ist das Verhalten Ihrer *checked* [illegible]eption anders als bei der originalen *unchecked* Variante?

Führen Sie das Resultat (UnitTests) dem Betreuer vor und erklären Sie im Code die wesentlichen Merkmale (Grenzfälle, ...) und Anpassungen.

2. Printer-Threads: Verwendung von Java Threads

Nachfolgend einige Basisübungen zum Starten und Stoppen von Threads in Java.

```
class PrinterThread extends Thread {
      char ch;
      int sleepTime;

      public PrinterThread(String name, char c, int t) {
            super(name);
            ch = c; sleepTime = t;
      }

      public void run() {
            System.out.println( getName() + " run laueft an");
            for (int i = 1; i < 100; i++) {
                  System.out.print (ch);
                  try { Thread.sleep(sleepTime);
```

```
                } catch (Interru                h e) { ; }
            }
            System.out.println(                 ame() + " run  fertig");
        }
}

public class Printer1 {
        public static void                  tring[] arg) {
                PrinterThread a =  new PrinterThread("PrinterA", '.', 0);
                PrinterThread b = new PrinterThread("PrinterB", '*', 0);
                a.start();
                b.start();
                b.run(); //Wie kann das abgefangen werden?
        }
}
```

COMPLETED

a) Studieren Sie das Programm Printer1.java: Die Methode **`Thread.run()`** ist **`public`** und kann daher direkt aufgerufen werden. Erweitern Sie die Methode **`run()`** so, dass diese sofort terminiert, wenn sie direkt und nicht vom Thread aufgerufen wird.
Tipp: Was liefert die Methode **`Thread.currentThread()`** zurück?

- Den aktuell ausführenden Thread

COMPLETED

b) Schreiben Sie das Programm so um, dass die run-Methode über das Interface **`Runnable`** implementiert wird.
Führen Sie dazu eine Klasse **`Printer`** ein, die das Interface **`Runnable`** implementiert.
Starten Sie zwei Threads, so dass dieselbe Ausgabe entsteht wie bei a).
Verwenden Sie den untenstehenden Code für den Test ihrer Implementation:

```
public class Printer2 {
        public static void main(String[] arg) {
                Printer a = new Printer('.', 0);
                Printer b = new Printer('*', 0);
                Thread t1 = new Thread(a, "PrinterA");
                Thread t2 = new Thread(b, "PrinterB");
                t1.start();
                t2.start();
        }
}
```

Thread.yield() -> obwohl Fairness bei Java concurrency nicht gegeben ist wächst die gleicheit der Ausgaben stark an.

COMPLETED

c) Wie kann die erreichen, die Fairness erhöhen, d.h. der Wechsel zwischen den Threads häufiger erfolgt? Wirkt es sich aufs Resultat aus?

COMPLETED

d) Wie muss man das Hauptprogramm anpassen, damit der Main-Thread immer als letztes endet?

Mit Thread.join wird auf eine beendiung eines Threads gewartet.
der Main Thread joint also einfach die beiden Printerthread und wartet bis diese beendigt haben.

3. Konto-Übertrag

Wir haben in der Vorlesung gesehen, dass die folgende Konto-Klasse "thread-safe" ist, da alle public-Methoden als "synchronized" implementiert sind.

```
class Account { // implemented as a monitor
      private int id;
      private int saldo = 0;

      public Konto(int id, int initialSaldo) {
            this.id = id; this.saldo = initialSaldo;
      }
      public int getId() {
         return id;
      }
      public synchronized int getSaldo () {
         return saldo;
      }
      public synchronized void setSaldo(int amount) {
            this.saldo=amount;
      }
      public   synchronized void changeSaldo (int delta) {
            this.saldo += delta;
      }
}
```

Ein SW-Entwickler der noch nicht viel von Synchronisation gehört hat, implementiert aufbauend auf der Klasse `Konto` eine Operation für den Transfer eines Geldbetrages zwischen zwei Konti. Die Klasse `KontoUebertragThread` implementiert dazu die Methode `kontoTransfer` (siehe Testprogramm AccountTransferTest.java). Das Testprogramm erzeugt mehrere Threads, die teilweise auf denselben Konto-Objekten operieren.

```
class AccountTransferThread extends Thread {
    private Account fromAccount, toAccount;
    private int amount;
    public AccountTransferThread (Account fromAccount, Account toAccount,
                                  int amount) {
        this.fromAccount = fromAccount;
        this.toAccount = toAccount;
        this.amount = amount;
    }

    public void accountTransfer() {
        if (fromAccount.getSaldo() > amount) { //do not overdraw account
            toAccount.veraendereKontoStand(-betrag);
            toAccount.veraendereKontoStand(betrag);
        }
    }

    public void run() {
        for (int i=0; i<10000; i++) {
            accountTransfer();
        }
    }
}
```

Es ist möglich das die Methode unterbrochen wird und der Amount des einen Accountes unter die 0 grenze fällt, da dies aber schon überprüft wurde, führt es zu einem Fehler.

COMPLETED

Diese Methode `accountTransfer` ist nicht thread-safe! Wieso? Konstruieren Sie dazu mögliche verzahnte Abläufe der Threads, die zu inkonsistenten Resultaten führen.

COMPLETED

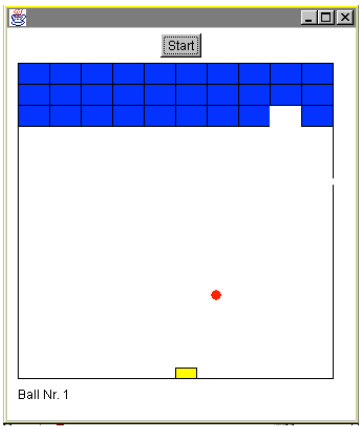
b) Erweitern Sie die Klasse Konto um eine sichere (thread-safe) Methode `accountTransfer`. Sie können diese Methode als Klassen- und oder Objekt-Methode implementieren. Passen Sie das Testprogramm entsprechend an.
Hinweis: Es macht Sinn, diese Methode in die Klasse `Account` zu integrieren, da dann die Synchronisation gekapselt ist und nicht Aufgabe des Verwenders der `Account`-Klasse ist.

EMERGENCY

c) Ändern Sie das Testprogramm so ab, dass ein Deadlock entsteht. Wie lassen sich Deadlocks generell vermeiden? Optional: Erweitern Sie ihre Implementation der `accountTransfer`-Methode so, dass Deadlocks nicht auftreten können.

4. Breakout: Thread Synchronisation

Breakout ist ein Reaktionsspiel, wo mit einem Schläger ein Ball geschlagen wird. Der Ball bewegt sich innerhalb eines rechteckigen Raumes, wobei eine Seite des Raumes aus drei Schichten von Ziegeln besteht:



Auf der gegenüberliegenden Seite befindet sich ein Schläger, der vom Benützer per Maus nach links und rechts bewegt werden kann. Wenn der Ball jeweils auf die Seitenwände oder auf die Ziegelmauer trifft, so prallt er daran ab. Bei der Ziegelmauer zerstört er dabei den getroffenen Ziegel. Auf der hinteren Seite verlässt der Ball den Raum, wenn er nicht vom Spieler mit dem Schläger zurückgeschlagen wird. Dazu muss der Spieler, den Schläger rechtzeitig an die richtige Position schieben.

Falls der Ball ein Loch in die Ziegelwand geschlagen hat und durch die Wand entweicht, hat der Spieler gewonnen. Falls der Spieler den Ball nicht trifft, geht der Ball verloren. Der Spieler erhält zu Beginn des Spiels fünf Bälle. Wenn er alle Bälle verloren hat, bevor ein Ball durch die Ziegelwand entweichen konnte, hat der Spieler das Spiel verloren.

Hinweise:

- Das Spiel besteht aus den folgenden Klassen:
  - **`Game`**: Enthält den Spielzustand und Statusfunktionen
  - **`GameView`**: Zeichnet das Spielfeld, Spielelemente und bedient die GUI Events

  - **`Brick`**: Repräsentiert einen Ziegelstein (z.B. ob schon getroffen)
  - **`Bat`**: Repräsentiert den Schläger, der vom Benutzer verschoben bedient wird.
  - **`Breakout`**: Hauptprogramm, welche das Spiel initialisiert und startet.
- Die Zeichenfunktionen und GUI Event-Loop sind im abgegebenen Gerüst bereits enthalten.
- Das Game-Objekt bietet verschiedene Funktionen der Form **`boolean hasHit`*`E`*`(int x, int y)`** um abzufragen, an einer bestimmten x/y-Koordinate ein Element berührt wird (je nach Element wird im Erfolgsfall auch der Status des Spiels verändert), wobei *`E`* sein kann:
  - **`Brick`**: Ziegel ist getroffen und wird zerstört markiert.
  - **`Bat`**: Schläger ist getroffen
  - **`Wall`**: eine Seitenwand wurde getroffen
  - **`BreakoutWall`**: Zielwand wurde getroffen und Spiel ist gewonnen
  - **`LostWall`**: Ball hat die Grundlinie berührt und ist verloren
- Das Game-Objekt kann auch gefragt werden, ob das Spiel läuft (gestartet und nicht gewonnen oder verloren) → **`boolean isRunning()`**

Aufgaben:

a) Der Ball soll in einem eigenen Thread laufen. Ergänzen Sie die Klasse Ball entsprechend. Solange der Ball im Spiel ist, soll er im run-loop alle 20ms mit **`updatePosition()`** auf die nächste Position bewegt werden. Andernfalls (z.B. Ball verloren) wird der run-loop und damit der Thread beendet. Ergänzen sie die Methode **`updatePosition()`**, so dass der Ball seine nächste Position berechnet, checkt, ob er irgendwo dagegen gestossen ist und gegebenen falls reagiert (z.B. Richtung wechseln, Ball verloren). Testen Sie jeweils auch, ob das Spiel noch läuft; falls nicht kann der Ball-Thread beendet werden.
   Erweitern Sie die Methode **`Game.startBall()`** damit ein Ball-Thread erzeugt und gestartet wird. Läuft das Spiel?
   i. Wo muss gegebenenfalls synchronisiert werden, damit keine Konflikte entstehen.
   ii. Was können Sie ändern, damit das Spiel flüssiger wird.

b) Ergänzen Sie **`Game`**, damit auch mehrere Bälle (gleichzeitig) gestartet werden können.
   i. Was müssen Sie erweitern, dass die Verwaltung der Threads klappt?
   ii. Sind gegebenen falls zusätzliche Synchronisationsmassnahmen notwendig.

Führen Sie das Endresultat dem Betreuer vor.