Praktikum Programmiermethodik 2 (Technische Informatik)
WS 2015/2016, Hochschule für Angewandte Wissenschaften (HAW), Hamburg
Prof. Dr. Philipp Jenke, Kasperczyk-Borgmann

Für dieses Aufgabenblatt gelten die folgenden Regeln:

- Der Java Code Style (siehe EMIL) ist einzuhalten. Es werden keine Abgaben abgenommen, die diese Anforderungen nicht erfüllen.
- Alle Programme für dieses Aufgabenblatt müssen in dem Package `aufgabenblatt2` liegen.
- Pro Team muss ein gemeinsames Git-Repository für das Projekt vorhanden sein (z.B. auf GitHub, Bitbucket oder dem HAW-Informatik-Home-Verzeichnis).
- Der Code muss vollständig getestet sein.

Aufgabenblatt 2: Lambdas, Streams, Threads

Aufgabe 2.1: Lambdas
Lernziele: Definition und Verwenden funktionaler Interfaces (SAM), Definition und Anwenden von Lambda-Ausdrücken.
Aufgabe:

- Schreiben Sie eine Klasse `Rechner`, die die vier Grundrechenarten beherrscht. Dazu bietet `Rechner` die Methode `double berechne(Operation, double, double)`.
- `Operation` ist ein `Enum` mit Konstanten für die vier Grundrechenarten.
- Intern verwaltet Rechner die vier Rechenoperationen in einer `Map`, wobei die Konstanten aus dem `Enum` die Schlüssel sind.
- Die Werte der `Map` sind jeweils Lambdas vom Typ `BinaryOperator`.
- In `berechne()` wird über die Operation der richtige Lambda-Ausdruck gewählt, auf die Argumente angewendet und das Ergebnis zurückgeliefert.
- Definieren Sie ein eigenes funktionales Interfaces (SAM) `DoubleDoubleZuDouble` mit der Methode `werteAus()`. Die Methode soll zwei Doubles als Argumente akzeptieren und als Ergebnis einen Double-Wert zurückliefern.
- Testen Sie das Interface mit mindesten zwei Lambda-Ausdrücken (z.B. Multiplikation + Nullstelle der Gleichung y=ax+b berechnen)

Aufgabe 2.2: Streams
Lernziele: Streams erzeugen, verarbeiten und terminieren
Aufgaben:

- Gegeben ist ein String-Array mit Benutzereingaben, die korrigiert werden sollen. Schreiben Sie dazu eine Verarbeitung mit meiner Streams-Kette, die folgende Funktionalität bietet:
  - Entfernen von `null`-Eingaben
  - Abschneiden der Leerzeichen am Anfang und Ende
  - Konvertierung von Klein- zu Großbuchstaben
  - Ersetzen Ä→AE, Ö→OE, Ü→UE, ß→SS
  - Kürzen der Strings auf maximal 8 Zeichen
- Im Ergebnis sollen die Strings in einer `List<T>` vorliegen.
- Beispiel:

{"Eingabe ", "Äußeres ", null, "Strassen-Feger", " ein Haus" } → [EINGABE, AEUSSERE, STRASSEN, EIN HAUS]

Aufgabe 2.3: Threads
Lernziele: Threads erzeugen, beenden, Interrupt-Ereignisse erzeugen und verarbeiten.
Aufgaben: Schreiben Sie einen kleinen Simulator für ein Autorennen.
Rennauto: Zunächst benötigen Sie ein Klasse `Rennauto`. Diese ist als Thread umgesetzt. Zur Laufzeit bewegt sich das Rennauto sekundenweise in Schritten weiter. Dazu hat ein Rennauto eine Durchschnittsgeschwindigkeit in m/s und kennt die Länge der Rennstrecke. Da die Fahrer mal besser und mal schlechter fahren, benötigen Sie aber nicht genau eine Sekunde pro Schritt. Stattdessen variiert die Zeit für einen Schritt zufallsbasiert zwischen 0.8 und 1.2 Sekunden. Nach jedem Schritt gibt das Auto seine aktuelle Position aus. Hat das Auto das Streckenende erreicht, endet der Thread. Das Auto merkt sich am Streckenende die benötigte Zeit in Sekunden (siehe `System.currentTimeMillis()`).

Rennen: Starten Sie nun mehrere Rennautos und lassen Sie sie ein Rennen fahren. Wenn alle Rennwagen im Ziel sind, dann soll das Rennende auf der Konsole bekannt gegeben werden und die Autos werden in der Reihenfolge ihrer Zielankunft aufgelistet. Beispiel (Streckenlänge: 10m, Durchschnittsgeschwindigkeit: 1m/s):

Wagen 1: 1,0/10,0
Wagen 0: 1,0/10,0
Wagen 2: 1,0/10,0
Wagen 1: 2,0/10,0
Wagen 0: 2,0/10,0
[...]
Wagen 1: 10,0/10,0
Wagen 2: 10,0/10,0
Wagen 0: 10,0/10,0
Rennen zuende.
Ergebnis:
Wagen 1: 9,7 sek.
Wagen 2: 10,4 sek.
Wagen 0: 10,6 sek.

Rennabbruch: Simulieren Sie nun zusätzlich einen Rennabbruch. Implementieren Sie dazu einen zusätzlichen Thread, der einmal pro Sekunde mit einer gewissen Wahrscheinlichkeit (z.B. 10%) einen Rennabbruch auslöst. Dazu muss der Rennabbruch-Thread die Rennwagen kennen und diesen bei einem Abbruch ein Interrupt senden.

Hinweis: Am Ende des Programms müssen alle Threads beendet sein. Sie dürfen dazu nicht `System.exit()`verwenden! Threads enden also entweder, weil auf natürlichem Wege das Ende der `run()`-Methode erreicht wurde, oder weil ein Interrupt empfangen wurde.