Check und Prepare

Vorbereitende Übungen

TECHNISCHE UNIVERSITÄT DARMSTADT

Autoren: Nhan Huynh und Darya Nikitina
Fachbereich: Informatik
Übungsblatt: 07
Version: 28. März 2022
Semesterübergreifend

V1 Theoriefragen

★★★

V1.1 Grundlegendes

1. Wie hängen die Begriffe `throws` und `throws` zusammen? Wo wird was verwendet?
2. Ist es sinnvoll, eigene Exceptionklassen zu definieren? Welche Vorteile ergeben sich hieraus?
3. Nennen Sie die Methoden der `Assert`-Klasse, die Sie zum Testen bei einem typischen JUnit Testcase in der Vorlesung kennengelernt haben, und beschreiben Sie kurz deren Verwendung.

V1.2 Wahr oder falsch?

Welche der folgenden Aussagen ist wahr?

(A) Auf einen `try`-Block muss immer mindestens ein `catch`-Block folgen.

(B) Wenn Sie eine Methode schreiben, die eine Exception auslösen könnte, müssen Sie diesen riskanten Code mit einem `try/catch`-Block umgeben.

(C) Auf einen `try`-Block können beliebig viele verschiedene `catch`-Blöcke folgen.

(D) Eine Methode kann nur eine einzige Art von Exception werfen.

(E) Die Reihenfolge der `catch`-Blöcke ist grundsätzlich gleich gültig.

(F) Laufzeit (Runtime)-Exceptions müssen gefangen oder deklariert werden.

(G) Es darf kein Code zwischen dem `try`-Block und dem `catch`-Block geschrieben werden.

(H) Eine Methode wirft eine Exception mit dem Schlüsselwort `throws`.

V2 Try/Catch-Block

★☆☆

Was ist das Problem mit dem folgenden Codeausschnitt?

```
1 public static void main(String[] args) {
2   int[] arr = new int[10];
3   System.out.println(arr[77]);
4 }
```

Modifizieren Sie den Code mittels `try/catch`-Blockes um das Problem zu beheben. Im `catch`-Block soll die Fehlerbotschaft mit der Methode `System.out .print()` auf der Konsole ausgegeben werden.

Übungsblatt 07: Vorbereitende Übungen von Nhan Huynh und Darya Nikitina

V3 Exceptions

Sehen Sie sich den folgenden Code genau an (`ExplodeException` erbt von `Exception`).

(1) Welche Ausgabe wird dieses Programm beim Aufruf der Methode `test()` liefern?

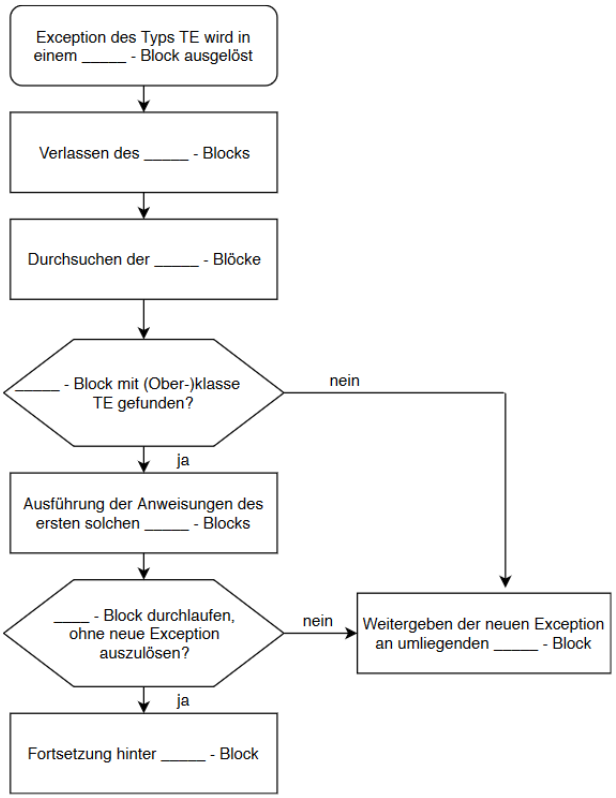
(2) Welche Ausgabe erfolgt bei einer Änderung von Zeile 2 in `String test = “yes“;`

```
1  public void test() {
2    String test = "no";
3    try {
4      doRisky(test);
5    } catch (ExplodeException ex) {
6      System.out.println("catching ExplodeException!");
7    }
8  }
9
10 public void doRisky(String test) throws ExplodeException {
11   System.out.println("begin doRisky");
12   if (test.equals("yes")) {
13     throw new ExplodeException();
14   }
15   System.out.println("end doRisky");
16   return;
17 }
```

V4 Ablaufdiagramm

In dieser Aufgabe beschäftigen wir uns mit dem Auffangen einer hypothetischen `Exception` des Typs `TE`. Ergänzen Sie in nebenstehenden Ablaufdiagramm an freien Stellen, ob es sich um einen `catch`- oder `try`-Block handelt

Ein Beispiel für Ablaufdiagramme finden Sie beispielsweise hier:
https://de.wikipedia.org/wiki/Programmablaufplan#Beispiel



V5 assert-Anweisungen

In der Vorlesung haben Sie die assert-Anweisungen kennengelernt.

1. Beschreiben Sie in eigenen Worten, was ein Assertion-Error ist.
2. Schreiben Sie den nachfolgenden Codeschnipsel kompakter mittels assert-Anweisungen!

```
1 if ((k > 0 && k + 1 <= 5) || (k % 3 == 2)) {
2   throw new AssertionError("Very bad k!");
3 }
```

3. Sie wissen, dass `assert`-Anweisungen beim Kompilieren an- oder abgeschaltet werden können mit entsprechenden Setzungen für den Compiler. Welche Vorteile ergeben sich hieraus? Warum sollten wir sie ausschalten und nicht einfach auch im realen Einsatz des Programms immer eingeschaltet mitlaufen lassen?

Information: assert-Anweisungen sind standardmäßig nicht angeschaltet. Um diese verwenden zu können muss der Flag `-eq` für die JVM[a] hinzugefügt werden.

[a] Java Virtual Machine

V6 Erster Test mittels BeforeEach

In dieser Aufgabe wollen wir einen Blick auf die `BeforeEach`-Annotation von JUnit 5 werfen. Methoden mit dieser Annotation werden **vor Beginn jedes einzelnen Tests** ausgeführt!

Gegeben sei eine Bibliothek für Geometrie-Funktionen. Dabei betrachten wir nur die Funktion `triangleArea`, die den Flächeninhalt eines Dreiecks berechnet und dazu die Längen der einzelnen Seiten (`a`, `b` und `c`) als `int`-Werte übergeben bekommt.

V6.1 Setup vor jedem Test

Gegeben sei folgende Klasse `GeoLib`:

```
1 public class GeoLib {
2
3   public GeoLib() {
4   }
5 }
```

Um die Funktionen der Bibliothek verwenden zu können, muss zunächst ein Objekt vom Typ `GeoLib` erzeugt werden. Hierzu können Sie den parameterlosen Standard-Konstruktor der Klasse `GeoLib` verwenden. Schreiben Sie eine entsprechend mit JUnit-Annotationen versehene Methode namens `setup`, die in einer Testklasse steht und die für jeden Testfall eine neue Instanz von `GeoLib` in dem bereits deklarierten Attribut `geoLib` speichert.

V6.2 Testfall

Schreiben Sie mindestens drei JUnit-Tests, die überprüfen, ob die Methode `triangleArea` für verschiedene Dreiecke korrekt arbeitet. Mindestens ein Testdreieck sollte dabei auch entartet sein.

Übungsblatt 07: Vorbereitende Übungen von Nhan Huynh und Darya Nikitina

V7 Fehlertypen

In der Vorlesung haben Sie kennengelernt, dass man Fehler in Programmen in zwei Kategorien einteilen kann. Es wurde unterschieden zwischen Kompilierzeitfehlern und Laufzeitfehlern.

(1) Beheben Sie im folgenden Codeausschnitt alle Kompilierzeitfehler:

```
1  public static int[] reverseArray(int[] source) throw Exception {
2    int length = source.length();
3    int[] inverted;
4    int i = 0;
5    int j = length;
6
7    try {
8      inverted = new int[length];
9
10     while (i < length) {
11       inverted[i] = source[j];
12       i--;
13       j++;
14     }
15   } catch (IndexOutOfBoundsException e) {
16     System.out.println("Caught Exception: " + e);
17   } catch (Exception) {
18     System.out.println("Caught Exception ...");
19   }
20
21   inverted = new int[length];
22   inverted = source;
23
24   return inverted;
25 }
```

(2) Was passiert generell beim Aufruf der Methode `reverseArray`? Warum kann der Code auch ohne vorhandene Kompilierzeitfehler nicht fehlerfrei ausgeführt werden? Was müsste man beheben um den Code ausführbar zu machen?

(3) Das Programm läuft zwar jetzt fehlerfrei, das Ergebnis entspricht aber noch nicht dem gewünschten Ergebnis (Array soll umgedreht werden). Beheben Sie alle fehlerhaften Stellen im Code, um das gewünschte Ergebnis zu erreichen.

V8 Testen mit JUnit - Qualitätskontrolle ★★★

Wir wollen ein neues System zur Qualitätskontrolle in einer Produktionskette testen. Hierzu gibt es eine Klasse `ProductLineManagement`, die Güter (Typ `Product`) herstellt. Ihre Tests sollen nun prüfen, ob dies schnell genug und hinreichend gut erfolgt. Die Maschinen garantieren dabei immer eine Mindestqualität von 89 ($= 89\%$ der optimalen Qualität).

Die Qualität wird gemessen auf einer Skala von 0 (defekt) bis 100 (perfekt).

Die Testklasse deklariert ein Attribut `static ProductLineManagement plm`, auf das Sie zugreifen können.

V8.1 Setup-Methode

Vor jedem Test muss die (sehr komplexe) Produktionskette initialisiert werden. Dies erfolgt durch den Aufruf des Konstruktors der Klasse `ProductLineManagement` mit dem Namen der Firma als `String`. Den Firmennamen dürfen Sie beliebig wählen. Geben Sie eine mit JUnit-Annotationen versehene öffentlich sichtbare Methode an, die diese Initialisierung vor jedem Test durchführt.

V8.2 Normalfall

Schreiben Sie einen Test für eine normale Produktion. Hierbei soll ein einziger Artikel `normalProduct` durch die Methode `Product produce (String)` der Klasse `ProductLineManagement` (siehe oben) produziert werden. Stellen Sie sicher, dass der gelieferte Artikel nicht `null` ist und eine Mindestqualität - abfragbar via `getQuality()` - von 89 besitzt. Als Titel des Artikels können Sie einen beliebigen `String` angeben.

V8.3 Behandlung von Exceptions

Schreiben Sie nun einen weiteren Test, der auch wie im vorherigen Aufgabenteil ein neues Produkt erstellt. Nur diesmal reichen Sie dieses Produkt mittels `boolean submit(Product, int)` aus der Klasse `ProductLineManagement` für die Qualitätskontrolle ein. Wurde die gewünschte Qualität erreicht, liefert die Methode `true`, andernfalls wirft die Methode eine `InsufficientQualityException`.

Testen Sie das Verhalten und das Auftreten der Exception mit einem Produkt, indem Sie dieses einmal auf die (garantierte) Mindestqualität von 89 und einmal auf die unerreichbare Qualität von 101 testen.

V9 Testen: Racket und Java ★★★

Sie haben nun sowohl das Testen in Java mittels JUnit, als auch das Testen in Racket mittels Checks kennengelernt. In dieser Aufgabe sollen Sie zuerst eine Problemstellung in beiden Sprachen lösen und anschließend Ihre Implementierungen testen.

Gegeben ist eine Zahlenliste. In Racket ist diese als Liste von `numbers` gegeben, in Java als Array von Typ `int[]`. Außerdem sind zwei Parameter `lower` und `upper` gegeben. Ziel ist es, alle Werte aus der Zahlenliste zu sortieren, welche nicht zwischen diesen beiden Grenzwerten `lower` und `upper` liegen (jeweils exklusive).

Ergänzen Sie die beiden untenstehenden Codeausschnitte und Verträge, um diese Problemstellung zu lösen.

```
;; Type: (list of number) number number -> (list of  number)
;; Returns:
(define (numbersBetween alon lower upper)
  ..... )
```

```
/**
 * @param a
 * @param lower
 * @param upper
 * @return
 */
public int[] betweenNumbers(int[] a, int lower, int upper) {
  .....
}
```

Sollte der Parameter `lower` dabei größer als der Parameter `upper` sein, so soll in Java eine `LowerBiggerThanUpperException` geworfen werden. Ergänzen Sie dies in Ihrer Implementierung.

In Racket haben Sie die Möglichkeit einen Fehler auszulösen. Dies geschieht über den Befehl (`error msg`), wobei `msg` ein `String` mit der gewünschten Fehlermeldung ist. Konventionsmäßig einigen wir uns darauf, dass wir bei `msg` zuerst den Funktionsnamen nennen, gefolgt von einem Doppelpunkt und einer Beschreibung des Fehlers. Lösen Sie äquivalent zur `LowerBiggerThanUpperException` auch in der Racketfunktion einen Fehler für diesen Fall aus.

Testen Sie abschließend die beiden Implementierungen mit jeweils 3 Tests. Ein Test sollte dabei das korrekte Werfen der Fehlermeldung testen.

V10 Exceptionklassen

V10.1

Schreiben Sie eine `public`-Klasse `MyException`, die von `Exception` erbt. Der Konstruktor dieser Klasse hat einen Parameter `str` vom Typ `String` und einen Parametern vom Typ `int`. Ein Objekt von `MyException` hat ein `private`-Attribut `message` vom Typ `String`. Der Konstruktor weist `message` die Konkatenation aus beiden Parametern zu. Die `public`-Methode `getMessage` von `Exception` soll so überschrieben werden, dass `message` zurückgeliefert wird.

V10.2

Schreiben Sie eine `public`-Klasse `X` mit einer `public`-Klassenmethode `km`, die einen `int`-Parameter `n` hat, `int` zurückliefert und potentiell `MyException` wirft. Und zwar wirft `km` eine `MyException` mit `"n cannot be negative"` und `n` als Parameterwerten, wenn negativ ist. Andernfalls liefert `km` das Quadrat von `n` zurück.

V10.3

Schreiben Sie eine `public`-Klasse `Y` mit einer `public`-Objektmethode `m`, die einen `int`-Parameter `n` hat und `int` zurückliefert. Diese Methode ruft `km` von `X` mit `n` auf, ohne ein Objekt von `X` dafür einzurichten, und liefert das Ergebnis von `km` zurück. Sollte `km` eine `Exception` werfen, dann soll die Botschaft der `Exception` auf dem Bildschirm ausgegeben werden.

V11 Welcher Belag darf es sein?

Schreiben Sie eine Klasse `NoBreadException` welche von `Exception` erbt, im Konstruktor einen Parameter `String topping` erhält und damit den Konstruktor der Basisklasse mit der Konkatenation `"There is no bread, only" + topping` aufruft.

Schreiben Sie dann ein Functional Interface namens `Lunch`. Dieses enthält die funktionale Methode `String getTopping(String s)`, welche eine `NoBreadException` wirft.

Initialisieren Sie nun das Functional Interface `Lunch` durch einen Lambda-Ausdruck. Geprüft werden soll, ob der `String` ein korrektes Sandwich ist. Dabei besteht ein korrektes Sandwich aus zweimal dem Substring `"bread` und einem Topping dazwischen. Korrekte Sandwiches sind also beispielsweise `"breadtunabread"` oder `"breadabcdefgbread"`. Vordem ersten `"bread"` und nach dem zweiten `"bread"` darf kein Substring mehr stehen. Zurückgegeben werden soll immer das Topping, also der Substring zwischen den beiden `"bread"`'s. Das Topping muss dabei nicht „sinnvoll" sein, sondern irgendein beliebiger `String`. Ist kein Brot vorhanden, so soll eine `NoBreadException` mit dem alleinigen Topping geworfen werden.

Sie dürfen davon ausgehen, dass niemals nur eine Brotscheibe verwendet wird, sondern entweder zwei oder keine.

V12 Arrays, Exceptions und Vererbung ★★★

V12.1 Klasse X

Gegeben sei die folgende Klasse:

```
1 public class X {
2
3   public int[] a;
4   public boolean[] writable;
5 }
```

Wir benutzen das Array, auf das `a` verweist, um `int`-Werte zu speichern. Im Array, auf das `writable` verweist, wird festgehalten ob ein `int`-Wert in `a` überschrieben werden darf, oder nicht. Der `int`-Wert `a[i]` darf überschrieben werden, wenn `writable[i] == true`. Sie können davon ausgehen, dass beide Arrays der Klasse `X` immer die gleiche Länge besitzen, sodass die Indizes der beiden Arrays übereinstimmen.

Implementieren Sie den Konstruktor der Klasse `X`, dieser bekommt einen `int`-Parameter übergeben und initialisiert die Arrays `a` und `writable` mit der gleichen Länge, dabei soll jeder Wert im Array `writable` mit `true` initialisiert werden. Die Länge beider Arrays entsprechen hier dem Wert des übergebenen Parameters. Erweitern Sie nun die Klasse um eine `public`-Methode `save`. Die Methode liefert nichts zurück, bekommt einen `int`-Parameter übergeben und speichert den übergebenen Parameter am kleinsten freien Index im Array `a` ab, an dem ein Wert nach aktueller Definition von `writable` überschrieben werden darf. Zusätzlich setzt sie den entsprechenden Wert im Array `writable` auf `false`. Darf kein Wert überschrieben werden, so soll die Methode eine `ArrayStoreException` mit der Nachricht `"no free space left"` werfen.

V12.2 Klasse Y

Schreiben Sie nun eine Klasse `Y`, die von der Klasse `X` aus Aufgabe rbt. Die Klasse soll die Methode `save` der Oberklasse überschreiben. Die Methode liefert nichts zurück, bekommt einen `int`-Parameter übergeben und soll mit diesem Parameter die Methode `save` der Oberklasse aufrufen. Wird dabei eine `ArrayStoreException` geworfen, so soll diese abgefangen werden. Ist dies der Fall, so sollen die beiden Arrays `a` und `writable` um ihre aktuelle Länge erweitert werden, um dann anschließend den übergebenen Parameter mittels der Methode `save` abspeichern zu können. Die bereits gespeicherten Werten in den beiden Arrays dürfen bei der Erweiterung nicht verloren gehen.