Funktionale und objektorientierte Programmierkonzepte Übungsblatt 08

Prof. Karsten Weihe

Übungsblattbetreuer:	Nick Steyer
Wintersemester 22/23	v1.0-SNAPSHOT
Themen:	`assert` und Exceptions
Relevante Foliensätze:	05 (und natürlich auch weiterhin 01*-04*)
Abgabe der Hausübung:	23.12.2022 bis 23:50 Uhr

Hausübung 08 — Gesamt: 25 Punkte

Excéptions – Gotta catch 'em all!

Beachten Sie die Seite *Verbindliche Anforderungen für alle Abgaben* in unserem Moodle-Kurs.

Verstöße gegen verbindliche Anforderungen führen zu Punktabzügen und können die korrekte Bewertung Ihrer Abgabe beeinflussen. Sofern vorhanden, müssen die in der Vorlage mit `TODO` markierten `crash`-Aufrufe entfernt werden. Andernfalls wird die jeweilige Aufgabe nicht bewertet.

Die für diese Hausübung relevanten Verzeichnisse sind `src/main/java/h08` und ggf. `src/test/java/h08`.

Einleitung

In diesem Übungsblatt werden Sie sich ausgiebig mit dem Thema Exceptions beschäftigen, das in der Vorlesung bereits theoretisch behandelt und in den vergangenen Übungsblättern kurz angerissen wurde. Sie werden zu Beginn eine simple Funktion schreiben, die Zahlen in einem Array addiert und die Summe ausgibt. Anschließend werden Sie sehen, was bei einer solch simplen Funktion alles schief gehen kann.

In der modernen Softwareentwicklung sollten Sie jede Funktion, die Sie implementieren, zusammen mit ihrem Namen und den formalen Parametern als eine Art Vertrag betrachten. Sagt der Name der Funktion aus, dass Zahlen addiert werden, dann sollte Ihre Funktion auch genau das tun – und nicht irgendetwas anderes. Kann der Vertrag nicht eingehalten werden, beispielsweise durch fehlerhafte aktuale Parameter, dann sollte diese Funktion eine Exception werfen. Sie sollte den Fehler *nicht* einfach ignorieren und z.B. 0 zurückgeben, wenn die Addition fehlschlägt. Durch den Wurf einer Exception kann der Fehler von der aufrufenden Methode sauber behandelt werden. Ignoriert die Methode die geworfene Exception, wird sie an die nächste aufrufende Methode weitergereicht – so ist es unmöglich, dass ein Fehler einfach ignoriert wird (wie das beim Rückgabewert 0 der Fall wäre). Außerdem werden so Regel- und Ausnahmefall besser getrennt.

Sie werden zunächst auf Exception-Klassen zurückgreifen, die bereits im Java-Framework enthalten sind, wie die `NullPointerException`. Manchmal sind diese Klassen jedoch nicht spezifisch genug, um den Ausnahmefall genau genug zu beschreiben[1]. Daher werden Sie im weiteren Verlauf auch eigene Exception-Klassen implementieren, die speziell für Ihr Programm zugeschnitten sind. Sie werden außerdem ein kleines Framework schreiben, um den Wurf der Exceptions zu erleichtern und Ihren Code übersichtlicher zu gestalten.

[1] Es gibt in der Praxis auch den umgekehrten Fall, dass man spezifische Ausnahmen durch generische ersetzt, um einem potentiellen Angreifer keine Informationen über die Interna des Systems zu liefern.

Zum Schluss werden Sie die von ihrer Funktion geworfenen Exceptions in ihrem Programm abfangen und entsprechend darauf reagieren. Sie werden außerdem Möglichkeiten kennenlernen, wie man in JUnit-Tests auf geworfene Exceptions reagieren kann. Denn nicht nur das reguläre Verhalten einer Methode sollte auf Korrektheit getestet werden, sondern auch das Ausnahmeverhalten bei fehlerhaften Eingaben.

H1: Methode mit `RuntimeExceptions` ?? Punkte

H1.1: Berechnung der Summe ?? Punkte

Im Verzeichnis `src/main/java/h08/calculation` finden Sie die Klasse `ArrayCalculatorWithRuntimeExceptions`. Diese implementiert die Objektmethode `addUp` der Schnittstelle `ArrayCalculator`. Die Methode bekommt einen Parameter `theArray` vom formalen Typ „Array von Array von `double`“ sowie einen Parameter `max` vom formalen Typ `double` übergeben und liefert einen Wert vom formalen Typ `double` zurück. Im Folgenden ist der Array, auf den `theArray` verweist, der *Hauptarray* (*primary array*), und die Arrays, auf die die Komponenten des Hauptarrays verweisen, sind die *Einzelarrays* (*secondary arrays*).

Berechnen Sie innerhalb der Methode `addUp` die Summe aller `double`-Werte in allen Einzelarrays im Hauptarray `theArray` und liefern Sie das Ergebnis zurück.

H1.2: Prüfen der Ausnahmefälle ?? Punkte

Die Summe kann nur korrekt berechnet werden, wenn die Eingabedaten keine Unstimmigkeiten aufweisen. Beispielsweise würde es zu Problemen führen, wenn `theArray` gleich `null` wäre. Deshalb ist es nun Ihre Aufgabe, *vor* der Berechnung der Summe die Eingabedaten zu prüfen und für jeden Ausnahmefall eine Exception aus der Java-Standardbibliothek zu werfen. Die Exceptions sind jeweils von der Klasse `java.lang.RuntimeException` abgeleitet. Die folgenden vier Fälle sind zu prüfen:

1. Falls `theArray` gleich `null` ist, wird die Abarbeitung von `addUp` durch den Wurf einer `java.lang.NullPointerException` mit der Botschaft `"Primary array is void!"` beendet.

2. Falls der erste Fall nicht eintritt und falls mindestens ein Einzelarray gleich `null` ist, wird die Abarbeitung durch den Wurf einer `NullPointerException` mit der Botschaft `"Secondary array at <i> is void!"` beendet, wobei `"<i>"` durch den kleinste infrage kommende Index im Hauptarray ersetzt wird, an welchem sich der ungültige Einzelarray befindet.

3. Falls weder der erste, noch der zweite Fall eintritt und falls `max` negativ ist, wird die Abarbeitung durch den Wurf einer `java.lang.ArithmeticException` mit der Botschaft `"Upper bound is negative!"` beendet.

4. Falls keiner der ersten drei Fälle eintritt und falls es irgendeine Komponente `theArray[i][j]` gibt, deren Wert entweder negativ oder größer als `max` ist, wird die Abarbeitung durch den Wurf einer `ArithmeticException` mit der Botschaft `"Value at (<i>,<j>) is not in range!"` beendet, wobei `"<i>"` durch den kleinsten infrage kommende Index im Hauptarray und `"<j>"` durch den kleinsten für dieses i infrage kommende Index im entsprechenden Einzelarray ersetzt wird.

Nur wenn keiner der oben aufgelisteten Ausnahmefälle eintritt, wird die Summe berechnet und zurückgeliefert.

> **Hinweis:**
>
> Beide Exception-Klassen haben bekanntlich einen Konstruktor mit einem Parameter vom formalen Typ `String`, welcher die Botschaft der Exception bestimmt.

Verbindliche Anforderung:

Es dürfen im Code von `addUp` insgesamt maximal *vier* `throw`-Anweisungen verwendet werden. (Falls Sie Hilfsmethoden für die Implementation von `addUp` verwenden, bezieht sich diese verbindliche Anforderung auf den Code aller von Ihnen erstellten Methoden in Summe.)

Unbewertete Verständnisfrage:

Haben Sie in den Kopf von `addUp` eine `throws`-Klausel eingefügt? Was sagt der Compiler, wenn Sie diese entfernen?

H2: Eigene Exception-Klassen ?? Punkte

In dieser Aufgabe werden Sie eigene Exception-Klassen auf Basis gegebener Klassen erstellen. Die benötigten Dateien befinden sich im Verzeichnis `main/java/h08/preconditions`.

Leiten Sie die gegebene `public`-Klasse `ArrayIsNullException` direkt von `RuntimeException` ab. Erstellen Sie in der Klasse einen `public`-Konstruktor, der parameterlos ist und die Botschaft der Klasse `RuntimeException` auf `"Array is null!"` setzt. Hierfür ruft er den Konstruktor der abgeleiteten Klasse mit einem Parameter vom Typ `String` auf und übergibt die Botschaft als aktualen Parameter.

Hinweis:

Die Klassen `Exception` und `RuntimeException` besitzen mehrere Konstruktoren. Wählen Sie jeweils denjenigen aus, der lediglich einen Parameter vom Typ `String` erwartet.

Leiten Sie die gegebene `public`-Klasse `WrongNumberException` direkt von `Exception` ab. Erstellen Sie für die Klasse einen `public`-Konstruktor mit einem Parameter vom formalen Typ `double`. Rufen Sie innerhalb dieses Konstruktors den Konstruktor der abgeleiteten Klasse mit einem Parameter vom Typ `String` auf und übergeben Sie eine `String`-Repräsentation des aktualen `double`-Parameterwertes.

Hinweis:

Um aus einem Parameter vom Typ `double` eine `String`-Repräsentation zu erhalten, können Sie wie in Kapitel 03b, ab Folie 85 der FOP vorgehen. Verwenden Sie eine Konkatenation von Zeichenkette und Zahl, jedoch mit leerer Zeichenkette.

Leiten Sie die gegebene `public`-Klasse `AtIndexException` direkt von `RuntimeException` ab. Erstellen Sie für die Klasse einen `public`-Konstruktor mit einem Parameter vom formalen Typ `int`. Rufen Sie innerhalb dieses Konstruktors den Konstruktor der abgeleiteten Klasse mit einem Parameter vom Typ `String` auf und übergeben Sie die Zeichenkette `"Index: <i>"` (mit Leerzeichen nach dem Doppelpunkt), wobei `"<i>"` durch eine `String`-Repräsentation des aktualen `int`-Parameterwertes ersetzt wird.

Leiten Sie die gegebene Klasse `AtIndexPairException` direkt von `Exception` ab. Erstellen Sie für die Klasse einen `public`-Konstruktor mit zwei Parametern jeweils vom formalen Typ `int`. Rufen Sie innerhalb dieses Konstruktors den Konstruktor der abgeleiteten Klasse mit einem Parameter vom Typ `String` auf und übergeben Sie die Zeichenkette `"Index: (<i>,<j>)"` (mit Leerzeichen nach dem Doppelpunkt), wobei `"<i>"` und `"<j>"` durch `String`-Repräsentationen der beiden aktualen Parameterwerte ersetzt werden. `"<i>"` ist der erste Parameter, `"<j>"` der zweite.

H3: Eigenes Preconditions-Framework ?? Punkte

Es gibt in der Programmierung einige Ausnahmefälle, die immer wieder überprüft werden müssen. In einer objektorientierten Sprache wie Java kommt es z.B. häufig vor, dass ein an eine Methode übergebener Parameter auf `null` geprüft werden soll. Ist er `null`, wirft die Methode eine `Exception`, da die Methode nur korrekt arbeiten kann, wenn ein Wert ungleich `null` übergeben wurde.

Um häufig vorkommende Ausnahmefälle nicht jedes Mal von Hand prüfen zu müssen, gibt es Frameworks, die die Arbeit erleichtern. So könnte solch ein Framework beispielsweise eine Methode `checkNotNull` beinhalten, die einen Parameter auf `null` überprüft und gegebenenfalls eine `NullPointerException` wirft. Diese Methode kann dann nacheinander für alle zu prüfenden Parameter aufgerufen werden. Hierdurch wird der Code übersichtlicher als bei der manuellen Implementierung und Fehler werden vermieden. Sie haben im Folgenden die Aufgabe, ein solches Framework selbst zu implementieren und anzuwenden.

H3.1: Die Klasse `Preconditions` ?? Punkte

Im Verzeichnis `src/main/java/h08/preconditions` finden Sie die Klasse `Preconditions`.

Überprüfen Sie innerhalb der vier Klassenmethoden jeweils einen Ausnahmefall aus H1.2 anhand der übergebenen Parameter. Liegt ein Ausnahmefall vor, soll die Methode eine der vier selbstdefinierten Exceptions aus H2 werfen. Offensichtlich sind die vier Methoden in `Preconditions` und die vier Exception-Klassen aus H2 genau für die vier Ausnahmefälle aus H1.2 definiert:

- `checkPrimaryArrayNotNull` wirft eine `ArrayIsNullException` im ersten Ausnahmefall
- `checkSecondaryArraysNotNull` wirft eine `AtIndexException` im zweiten Ausnahmefall; übergeben Sie hier den Index des ungültigen Einzelarrays als aktualen Parameter an den Konstruktor der Exception
- `checkNumberNotNegative` wirft eine `WrongNumberException` im dritten Ausnahmefall; übergeben Sie hier `max` als aktualen Parameter an den Konstruktor der Exception
- `checkValuesInRange` wirft eine `AtIndexPairException` im vierten Ausnahmefall; übergeben Sie hier die beiden Indices des ungültigen Wertes als aktuale Parameter an den Konstruktor der Exception

Beispiel:

Die Methode `checkPrimaryArrayNotNull` überprüft, ob der übergebene Parameter `primaryArray` gleich `null` ist. Ist das der Fall, wird eine Exception vom Typ `ArrayIsNullException` geworfen.

Verbindliche Anforderungen:

Fügen Sie selbstständig benötigte `throws`-Klauseln zur Methode hinzu. Diese dürfen *nicht* einfach `Exception` oder `RuntimeException` deklarieren, sondern müssen die konkreten Exception-Klassen aus H2 explizit angeben.

Unbewertete Verständnisfragen:

- Sie haben jetzt zwei Möglichkeiten angewandt, wie man mit Ausnahmefällen umgehen kann:
  1. passende Runtime-Exception-Klassen
  2. passende selbstdefinierte Exception-Klassen, die entweder von `RuntimeException` oder direkt von `Exception` abgeleitet sind

  Was spricht für, was gegen jede dieser Optionen?
- Anstelle von `"Index: <i>"` hätte man in `AtIndexException` auch einfach nur `"<i>"` analog zu `WrongNumberException` zur Botschaft machen können. In einem echten Projekt hätten wir das auch so gemacht. Was spricht für die eine, was für die andere Variante?
- Was sagt der Compiler, wenn Sie die `throws`-Klauseln entfernen? Was sagt er, wenn Sie die vier von `Exception` abgeleiteten Klassen in den Klauseln durch eine `java.lang.Exception`-Klausel austauschen? Wie unterscheidet sich die `throws`-Klausel in H1.2 von den `throws`-Klauseln in dieser Aufgabe? Passen die Ergebnisse zu dem, wie Sie das Thema Exceptions und insbesondere Runtime-Exceptions in Kapitel 05 verstanden haben? *Vergessen Sie nicht, die Änderungen wieder rückgängig zu machen.*

H3.2: Verwendung des `Preconditions`-Frameworks ?? Punkte

Im Verzeichnis `src/main/java/h08/calculation` finden Sie die Klasse `ArrayCalculatorWithPreconditions`.

Berechnen Sie innerhalb der Methode `addUp` die Summe aller `double`-Werte in allen Einzelarrays im Hauptarray `theArray` und liefern Sie das Ergebnis zurück – genau wie bei H1.1.

Prüfen Sie *vor* der Berechnung der Summe mithilfe der eben implementierten `Preconditions`-Klasse die vier in H1.2 spezifizierten Ausnahmefälle ab. In H3.1 wird beschrieben, welche Methode der `Preconditions`-Klasse welchen Ausnahmefall abprüft.

Verbindliche Anforderung:

Fügen Sie selbstständig benötigte `throws`-Klauseln zur Methode hinzu. Diese dürfen *nicht* einfach `Exception` oder `RuntimeException` deklarieren, sondern müssen die konkreten Exception-Klassen aus H2 explizit angeben.

H4: Print-Methode ?? Punkte

Im Verzeichnis `src/main/java/h08` finden Sie die Klasse `Main` mit der Klassenmethode `print`.

Berechnen Sie mithilfe der Methode `addUp` der Klasse `ArrayCalculatorWithPreconditions` die Summe der Werte aller Einzelarrays im gegebenen Array `theArray`.

- Wenn bei der Berechnung kein Ausnahmefall auftritt, soll `"Sum: <i>"` (mit Leerzeichen nach dem Doppelpunkt) ausgegeben werden, wobei `"<i>"` durch eine `String`-Repräsentation der berechneten Summe ersetzt wird.
- Wird eine `AtIndexException` oder eine `AtIndexPairException` geworfen, so soll `"Bad array: <message>"` (mit Leerzeichen nach dem Doppelpunkt) ausgegeben werden, wobei anstelle von `"<message>"` die Botschaft der aufgetretenen Exception eingesetzt werden soll.
- Tritt eine `WrongNumberException` auf, so soll `"Bad max value: <message>"` (mit Leerzeichen nach dem Doppelpunkt) ausgegeben werden, wobei anstelle von `"<message>"` die Botschaft der aufgetretenen Exception

eingesetzt werden soll.

- Andere Ausnahmefälle werden nicht abgefangen.

Die Ausgabe erfolgt in allen Fällen mithilfe von `System.out.println`.

Verbindliche Anforderung:

Fügen Sie *keine* `throws`-Klauseln zur Methode hinzu.

Unbewertete Verständnisfrage:

Wieso muss der Fall, dass eine `ArrayIsNullException` geworfen wird, nicht abgefangen werden? Was passiert, wenn eine solche Exception geworfen wird?

H5: Tests mit JUnit ?? Punkte

Zum Schluss werden Sie noch Tests mithilfe der Klasse `CalculatorTests` durchführen. Diese finden Sie im Verzeichnis `src/test/java/h08`. Die eigentlichen Testmethoden für die beiden Klassen `ArrayCalculatorWithRuntimeExceptions` und `ArrayCalculatorWithPreconditions` sind in der Klasse schon enthalten. Sie benutzen jedoch zwei Hilfsmethoden, die von Ihnen zu implementieren sind.

H5.1: Testen der Summenberechnung ?? Punkte

Die Methode `testSum` hat einen Parameter `sut` vom formalen Typ `ArrayCalculator`. `ArrayCalculator` ist eine Schnittstelle, die die Methode `addUp` beinhaltet und sowohl von `ArrayCalculatorWithRuntimeExceptions` als auch von `ArrayCalculatorWithPreconditions` implementiert wird. Rufen Sie innerhalb der Methode `testSum` die `addUp`-Methode auf dem gegebenen Objekt auf. Verwenden Sie hierfür die gegebenen Parameter `array` und `max`. Überprüfen Sie, ob das berechnete Ergebnis dem erwarteten Ergebnis `expectedSum` entspricht. Der Test muss fehlschlagen, wenn die berechnete Summe nicht dem erwarteten Ergebnis entspricht.

Ebenso muss der Test fehlschlagen, wenn bei der Berechnung der Summe eine Exception geworfen wird. In diesem Fall soll der vom Test geworfene `AssertionError` die Botschaft `"Unexpected exception: <message>"` (mit Leerzeichen nach dem Doppelpunkt) beinhalten, wobei `"<message>"` durch die Botschaft der unerwartet geworfenen Exception ersetzt wird.

H5.2: Test der Ausnahmebehandlung ?? Punkte

Hinweis:

Im JUnit Framework gibt es die Methode `assertThrowsExactly`. Diese Methode verifiziert, dass die Ausführung eines gegebenen Codes zum Wurf einer Exception führt. Wird keine Exception geworfen, so sorgt diese Methode dafür, dass der Test fehlschlägt. Ebenso schlägt der Test fehl, wenn der Datentyp der geworfenen Exception nicht dem erwarteten Typ entspricht. Wird eine Exception geworfen, so bildet diese den Rückgabewert der Methode.

Die Methode bekommt zwei Parameter übergeben[a]: Der erste ist vom formalen Typ `Class` und repräsentiert die Klasse der erwarteten Exception (genauer, des erwarteten `Throwable`-Objekts). Der zweite Parameter ist vom formalen Typ `Executable` und repräsentiert das Programm, das die Exception werfen soll. Hier können Lambda-Ausdrücke verwendet werden, wie Sie sie schon in Kapitel 04c, ab Folie 94 der FOP und in Übungsblatt H07 kennengelernt haben.

[a] Es gibt noch weitere Überladungen, die für diese Aufgabe jedoch nicht relevant sind.

Rufen Sie innerhalb der Methode `testException` die `addUp`-Methode auf dem gegebenen Objekt `sut` auf. Verwenden Sie hierfür die gegebenen Parameter `array` und `max`. Überprüfen Sie dabei mithilfe von `assertThrowsExactly`, ob der Aufruf zum Wurf einer Exception vom erwarteten Typ (`expectedException`) führt. Der Test muss fehlschlagen, wenn keine Exception geworfen wird oder der Datentyp der Exception nicht dem erwarteten Typ entspricht.

Ebenso muss der Test fehlschlagen, wenn die Botschaft der geworfenen Exception nicht der erwarteten Botschaft `expectedExceptionMessage` entspricht. In diesem Fall soll der vom Test geworfene `AssertionError` die Botschaft `"<expectedExceptionMessage> : <actualMessage>"` (mit Leerzeichen vor und nach dem Doppelpunkt) beinhalten, wobei `"<expectedExceptionMessage>"` durch die erwartete und `"<actualMessage>"` durch die tatsächliche Botschaft der Exception ersetzt werden.