

Funktionale und objektorientierte Programmierkonzepte

Übungsblatt 08



TECHNISCHE
UNIVERSITÄT
DARMSTADT

Prof. Karsten Weihe

Übungsblattbetreuer:
Wintersemester 22/23
Themen:
Relevante Foliensätze:
Abgabe der Hausübung:

Nick Steyer
v1.0-SNAPSHOT
assert und Exceptions
05 (und natürlich auch weiterhin 01*-04*)
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 in der Vorlage relevanten Verzeichnisse sind `src/main/java/h08` und `src/test/java/h08`.

Einleitung

H1: Methode mit `RuntimeExceptions`

?? Punkte

H1.1: Berechnung der Summe

?? Punkte

Im Verzeichnis `src/main/java/h08/calculation` befindet sich die Datei `ArrayCalculatorWithRuntimeExceptions.java`. Die in dieser Datei deklarierte Klasse 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* der kleinste infrage kommende Index im Hauptarray ist, 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* der kleinste infrage kommende Index im Hauptarray und *j* der kleinste für dieses *i* infrage kommende Index im entsprechenden Einzelarray ist.

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 erstellen. Die benötigten Dateien befinden sich im Verzeichnis `main/java/h08/preconditions`.

Leiten Sie die Klasse `ArrayIsNullException` in `ArrayIsNullException.java` direkt von `RuntimeException` ab. Erstellen Sie für die 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 aktuellen 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 Klasse `WrongNumberException` in `WrongNumberException.java` 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 aktuellen `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 Klasse `AtIndexException` in `AtIndexException.java` 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* eine `String`-Repräsentation des aktuellen `int`-Parameterwertes ist.

Leiten Sie die Klasse `AtIndexPairException` in `AtIndexPairException.java` 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* `String`-Repräsentationen der beiden aktuellen Parameterwerte sind. *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 objekt-orientierten 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` befindet sich die Datei `Preconditions.java` mit der 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 aktuellen Parameter an den Konstruktor der Exception
- `checkNumberNotNegative` wirft eine `WrongNumberException` im dritten Ausnahmefall; übergeben Sie hier `max` als aktuellen Parameter an den Konstruktor der Exception
- `checkValuesInRange` wirft eine `AtIndexPairException` im vierten Ausnahmefall; übergeben Sie hier die beiden Indices des ungültigen Wertes als aktuelle 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 Exception-Klassen in den Klauseln jeweils durch `java.lang.Exception` austauschen? Wie unterscheiden sich die `throws`-Klausel in H1.2 und die `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` befindet sich die Datei `ArrayCalculatorWithPreconditions.java` mit der 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` befindet sich die Datei `Main.java` mit der Klasse `Main` und 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 werden soll.
- 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. Sie finden Sie im Verzeichnis `src/test/java/h08` in der Datei `CalculatorTests.java`. 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 werden soll.

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.

^aEs 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 zu ersetzen sind.