Assertions und Annotations

Grundlagen zu Assertions

Mit **Assertions** (Behauptungen) können Sie während der Entwicklungszeit zu Testzwecken gezielt einzelne Programmzustände überwachen.

- Die Behauptung wird so als Bedingung (boolescher Ausdruck) formuliert, dass sie den Wert `true` liefert, wenn das Programm ordnungsgemäß abläuft.
- Trifft die Behauptung nicht zu, wird eine Exception `AssertionError` ausgelöst und das Programm wird sofort beendet. Sie können nun prüfen, wie der „fehlerhafte“ Programmzustand entstanden ist.
- Die Auswertung der Assertions wird bei der Ausführung des Interpreters eingeschaltet. Standardmäßig ist die Auswertung der Assertions ausgeschaltet.
- Assertions dürfen nicht eingesetzt werden, um einen korrekten Programmablauf zu sichern, da ihre Ausführung auch ausgeschaltet werden kann. Assertions sollten keine Seiteneffekte (side effects) bei der Auswertung der Ausdrücke auslösen, z. B. wenn zur Auswertung des Ausdrucks Methodenaufrufe notwendig sind, die den Zustand des Programms ändern.

Assertions einsetzen

Syntax für eine Assertion

- Eine Assertion beginnt mit dem Schlüsselwort `assert`.

```
assert expression1: stringExpression;
       ①            ②
```

- Es folgt ein Ausdruck, der einen Wert vom Typ `boolean` liefert ①.
- Wird dieser Ausdruck zu `true` ausgewertet, wird das Programm mit der nächsten Anweisung fortgesetzt.
- Liefert der Ausdruck den Wert `false`, wird eine `AssertionError`-Exception ausgelöst.
- Zusätzlich zu der Bedingung geben Sie mit einem Doppelpunkt `:` abgetrennt einen String-Ausdruck ② als erläuternde Fehlermeldung an. Der String wird der `AssertionError`-Exception als Parameter übergeben. Über die Methode `getMessage` kann der Text der so festgelegten Fehlermeldung beim Abfangen der Exception ausgewertet werden.
- Eine Variante der `assert`-Anweisung besitzt keinen speziellen Meldungstext ③.

```
assert expression2; ③
```

Beispiel: *com\herdt\java9\kap18\AssertTest.java*

Beim Start der Anwendung soll der Benutzer eine Zahl zwischen 10 und 50 eingeben, um die maximale Betriebstemperatur einer Maschine festzulegen.

- Wird ein Wert außerhalb des Intervalls eingegeben, wird die Temperatur auf den Standardwert 20 Grad gesetzt.
- Gibt der Benutzer keine Zahl, sondern beispielsweise Text ein, wird eine Exception ausgelöst. Allerdings wurde an dieser Stelle „vergessen“, den Initialwert von `i` (die Zahl 0) auf eine Temperatur im Intervall 10..50 (beispielsweise auf den Standardwert) zu setzen.
- Durch die zusätzliche Assertion am Ende der Methode wird dies bemerkt und eine entsprechende Meldung ausgegeben.

```
   public static void main(String args[])
   {
     final int TEMPERATURE_STANDARD = 20;
     int i = 0;
①    try
     {
       if (args.length > 0)
         i = Integer.parseInt(args[0]);
②      if (!((i >= 10) && (i <= 50)))
         i = TEMPERATURE_STANDARD;
     }
③    catch (NumberFormatException nfe)
     {
       System.out.println
     ("Als Parameter eine Zahl zwischen 10 und 50 eingeben.");
     }
④    assert ((i >= 10) && (i <= 50)):
⑤      "ASSERT: Die Zahl " + i + " lag nicht zwischen 10 und 50";
     System.out.println("Eingestellte Temperatur: " + i);
   }
```

① Der `try-catch`-Block fängt die Exception ab, die möglicherweise bei der Umwandlung des Programmparameters in eine Zahl erzeugt wird.

② Es wird geprüft, ob sich der übergebene Wert im Intervall 10...50 befindet. Ansonsten wird die Temperatur auf den vordefinierten Standardwert 20 gesetzt.

③ Wenn sich der Parameterwert nicht in eine Zahl umwandeln lässt, wird eine `NumberFormatException` ausgelöst. Bei der Exception-Behandlung wird hier allerdings die Temperatur nicht auf den Standardwert 20 gesetzt.

④ Am Schluss der Methode wird behauptet (Assertion), dass der Wert innerhalb des Intervalls liegt.

⑤ Wenn diese Behauptung nicht zutrifft (wenn die Temperatur nicht im Intervall liegt), löst die Assertion eine Exception (`AssertionError`) aus. Das Programm wird sofort mit einer entsprechenden Meldung mit dem vorgegebenen Text angehalten.

Sie haben die Möglichkeit, einen `AssertionError`, d. h. eine Exception, die durch eine `assert`-Anweisung ausgelöst wurde, mit einem `try-catch`-Block abzufangen und zu behandeln. Da dies den Programmcode unnötig erweitert, wird ein `AssertionError` üblicherweise nicht behandelt. Aussagekräftige Hinweise in der `assert`-Anweisung genügen gewöhnlich als Information.

Anwendungsmöglichkeiten für Assertions

<table>
<tr><td rowspan="2">Nachbedingung prüfen</td><td>Mit einer Assertion am Ende einer Methode kann beispielsweise geprüft werden, ob die Methode korrekte Ergebnisse liefert.</td></tr>
<tr><td><pre>
public void somethingToDo()
{
  boolean status = true;
  ...
  assert status:
    "Der Status darf sich nicht ändern.";
}
</pre></td></tr>
<tr><td rowspan="2">Nicht erreichbare Programmteile kennzeichnen</td><td>Mit einer Assertion können Sie überwachen, ob Programmteile, die nicht erreicht werden sollen, auch tatsächlich nicht ausgeführt werden.

Im nachfolgenden Beispiel ist vorgesehen, dass alle möglichen Werte für <code>value</code> in der <code>switch</code>-Anweisung mit <code>case</code> berücksichtigt werden. Sollte jedoch ein anderer Wert auftreten, so würde bei <code>default</code> die Assertion ausgelöst. Die Assertion enthält hier als Behauptung den Wert <code>false</code>. Sie wird daher immer ausgelöst, sobald der Programmteil erreicht wird.</td></tr>
<tr><td><pre>
switch (value)
  //nur die Werte 1 und 2 sollten auftreten
{
 case 1: .... break;
 case 2: .... break;
 default: assert false: value;
  //Als Text wird der Wert ausgegeben
}
</pre></td></tr>
</table>

Grundsätzlich haben Sie auch die Möglichkeit, eine **Vorbedingung** zu **prüfen**, d. h. eine Assertion am Anfang einer Methode zu verwenden, mit der Sie beispielsweise die Gültigkeit von Werten prüfen, die als Parameter übergeben wurden. Üblicherweise wird eine solche **Prüfung der Vorbedingung** jedoch nicht vorgenommen, da die Methode keinen Einfluss auf die korrekte Arbeitsweise der aufrufenden Methode hat. Bei der Programmentwicklung ist es hilfreicher, mit einer Nachbedingung zu prüfen, ob eine Methode korrekte Ergebnisse **liefert**, statt mit einer Vorbedingung den **Erhalt** sinnvoller Werte zu prüfen.

Assertions bei der Programmausführung auswerten

Standardmäßig ist die Auswertung von Assertions ausgeschaltet, d. h., die Anweisungen werden bei der Programmausführung ignoriert. Im Quelltext verwendete `assert`-Anweisungen werden aber in die **.class*-Datei kompiliert, unabhängig davon, ob sie später verwendet werden sollen. Um Assertions zu verwenden, müssen diese bei der Ausführung des Programms über den Interpreter angeschaltet werden (die Ausführungsgeschwindigkeit wird durch die Auswertung von Assertions nicht spürbar langsamer).

- Fügen Sie dem Befehl zur Ausführung des Programms die Option `-ea` (enable Assertions) hinzu:

  ```
  java -ea ClassName
  ```

  Auf diese Weise wird die Verwendung der Assertions für alle verwendeten Klassen angeschaltet.
- Es besteht weiterhin die Möglichkeit, die Verwendung von Assertions für bestimmte Klasse und Packages (einschließlich der SubPackages) ein- bzw. auszuschalten.
- Sie können mehrere Optionen zum Ein- bzw. Ausschalten der Assertions kombinieren.

`-ea, -da` **`-esa, -das`**	Über diese Parameter schalten Sie die Verwendung von Assertions für alle Klassen (ohne Systemklassen) ein (`-ea` – Kurzform für `-enableassertions`) oder aus (`-da` – Kurzform für `-disableassertions`). Um auch die Systemklassen einzubeziehen, verwenden Sie die Option `-esa`.
`-ea:ClassName` **`-da:ClassName`**	Die Verwendung von Assertions wird für die angegebene Klasse ein- bzw. ausgeschaltet.
`-ea:...` **`-da:...`**	Die Verwendung von Assertions wird für das Default-Package (das aktuelle Verzeichnis) ein- bzw. ausgeschaltet.
`-ea:PackageName...` **`-da:PackageName...`**	Die Verwendung von Assertions wird für das angegebene Package, einschließlich aller SubPackages, ein- bzw. ausgeschaltet.

Beispiel

```
① java -ea -da:MyClass -da:com.herdt.java9...
② java -ea:MyClass -ea:com.herdt.java9...
```

① Der erste Beispielbefehl schaltet die Verwendung von Assertions zunächst für alle Klassen ein und schränkt dies dann ein: Die Auswertung von Assertions wird für die Klasse `MyClass` und für das Package `com.herdt.java9`, inklusive aller SubPackages, ausgeschaltet.

② Die Verwendung ist standardmäßig ausgeschaltet. Deshalb wird der Schalter `-da` nicht explizit im zweiten Beispiel angegeben. Hier wird die Auswertung der Assertions für die Klasse `MyClass` und das Package `com.herdt.java9` einschließlich aller SubPackages eingeschaltet.

Annotations

Grundlagen zu Annotations

Annotations sind Metadaten zur Beschreibung des im Programm enthaltenen Programmcodes. Diese Metadaten werden selbst in den Programmcode eingefügt. Sie selbst liefern nur eine Information, deren Auswertung muss anderweitig erfolgen. Mögliche Komponenten zur Beschreibung mit Metadaten sind Klassen, Methoden, Felder, Parameter, Variablen, Konstruktoren, Pakete und seit Java 8 auch Typen.

Annotations sind seit Java 5 ein Sprachbestandteil von Java. Im Umfeld von Java EE finden sie verstärkt Einsatz, dienen dort aber auch Konfigurationszwecken.

Die Auswertung von Annotations erfolgt direkt durch den Compiler, durch spezielle Werkzeuge oder auch programmatisch durch sogenannte **Reflections**.

Syntax für eine Annotation

```
@<AnnotationName>
  [(Schlüssel1 = Wert1[, Schlüssel,2 = Wert2[, …]])] ①
```

- Eine Annotation beginnt mit dem Zeichen @.
- Danach folgt der Name der Annotation. Dieser wird großgeschrieben.
- Annotations können zusätzliche Informationen ① enthalten. Es wird unterschieden zwischen:
  - Marker-Annotations – diese enthalten keine zusätzliche Information. Die Klammern müssen hier nicht mit angegeben werden, beispielsweise
    `@Override`
  - Annotations mit einem Wert (Single-Value-Annotation). Der Schlüssel des Wertes wird im Regelfall `value` benannt. Ist dies der Fall, muss die Bezeichnung des Wertes nicht mit angegeben werden, beispielsweise
    `@SuppressWarnings("all")`
  - Annotations mit mehreren Werten (volle Annotation). Bei diesen werden die Informationen als Schlüssel/Wert-Paare gesetzt, beispielsweise
    `@IntegerSetter(min = 0, max = 5)`

Vordefinierte Annotationstypen

Das Package `java.lang` enthält vordefinierte Annotationstypen, die vom Compiler ausgewertet werden und Meldungen beim Compilieren erzeugen bzw. unterdrücken.

`@Deprecated`	Veraltete Programmbestandteile (Klassen, Methoden …) können mit dieser Annotation gekennzeichnet werden.
`@FunctionalInterface`	Kennzeichnet ein Interface als Funktionales Interfaces (Das Interface besitzt nur eine abstrakte Methode), ab Version 9
`@Override`	Die Kennzeichnung von Methoden einer Klasse, welche aus Basisklassen oder Interfaces geerbt und überschrieben werden sollen, erfolgt mit dieser Annotation.
`@SuppressWarnings`	Dient der Unterdrückung von Warnungen des Compilers.
`@SafeVarargs`	Diese Annotation wird für die Kennzeichnung von Methoden mit variabler Argumentzahl genutzt.

Weitere Annotationstypen, die zur Definition von Annotations benötigt werden, sind im Package `java.lang.annotation` enthalten.

Neben der Verwendung der vordefinierten Annotationstypen besteht die Möglichkeit, eigene Annotations zu erstellen. Der Aufbau einer Annotation entspricht dem eines Interface. Die Auswertung eigener Annotations erfolgt über sogenannte **Reflections**.

Beispiel: Anwendung von `@Override` zur Kennzeichnung einer überschriebenen Methode

com\herdt\java9\kap18\AnnotationOverrideSuper.java
com\herdt\java9\kap18\AnnotationOverride.java

Im Beispiel wird durch einen Fehler beim Festlegen des Methodennamens der zu überschreibenden Methode durch die verwendete Annotation `@Override` eine Fehlermeldung vom Compiler ausgelöst.

```
   package com.herdt.java9.kap18;
①  class AnnotationOverrideSuper
   {
     public void writeText()
     {
       System.out.println("Ausgabe der Superklasse");
     }
   }
```

① In der Klasse `AnnotationOverrideSuper` wird eine einfache Methode `writeText()` definiert.

```
   package com.herdt.java9.kap18;
①  class AnnotationOverride extends AnnotationOverrideSuper
   {
②    @Override
③    public void writeTex()
     {
       System.out.println("Ausgabe der abgeleiteten Klasse");
     }

     public static void main (String[] args)
     {
        AnnotationOverride instance = new AnnotationOverride();
        instance.writeText();
     }
   }
```

① Die Klasse `AnnotationOverride` wird von der Klasse `AnnotationOverrideSuper` abgeleitet.

② Die Annotation `@Override` gibt an, dass die folgende Methode eine Methode der Superklasse überschreibt.

③ Beim Namen der zu überschreibenden Methode wird ein Buchstabe 'vergessen'. Die Methode `writeTex()` ist nicht in der Superklasse vorhanden. Durch die Annotation wird beim Kompilieren ein Fehler erzeugt. Ein Auskommentieren der Annotation führt dazu, dass der Compilerlauf erfolgreich abgeschlossen wird. Die erzeugte Ausgabe heißt dann jedoch "*Ausgabe der Superklasse*" und nicht wie beabsichtigt "*Ausgabe der abgeleiteten Klasse*".

```
C:\uebung\jav9_bu\com\herdt\java9\kap18\AnnotationOverride.java:5
: error: method does not override or implement a method from a
supertype
  @Override
  ^
```

Meldung des Compilers durch den fehlerhaften Methodennamen

Beispiel: Anwendung von `@Deprecated` zur Kennzeichnung einer veralteten Methode

com\herdt\java9\kap18\AnnotationDeprecatedSuper.java
com\herdt\java9\kap18\AnnotationDeprecated.java

Im Beispiel wird mit der Annotation `@Deprecated` in der Oberklasse eine Methode als veraltet gekennzeichnet. Beim Kompilieren der Klasse, welche von dieser Klasse abgeleitet ist, wird zuerst nur eine einfache Meldung angezeigt. Gleichzeitig erfolgt der Hinweis, dass bei Anwendung des Schalters `-Xlint:deprecation` detailliertere Informationen geliefert werden.

```
Note:
C:\uebung\jav9_bu\com\herdt\java9\kap18\AnnotationDeprecated.java
uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.
```

Meldung des Compilers bei Standard-Compilierung

Nach einem erneuten Kompilieren mit dem Schalter wird eine detaillierte Meldung auf der Konsole ausgegeben.

```
C:\uebung\jav9_bu\com\herdt\java8\kap18\AnnotationDeprecated.java
:8: warning: [deprecation] writeText() in
AnnotationDeprecatedSuper has been deprecated
    instance.writeText();
             ^
1 warning
```

Compilermeldung bei Kompilierung mit dem Schalter `-Xlint:deprecation`

Übung

Assertions

Level		Zeit	ca. 10 min
Übungsinhalte	✔ Anwendung von Assertions		
Übungsdatei	--		
Ergebnisdatei	*Exercise.java*		

1. Einem Programm sollen zwei Parameter übergeben werden. Überprüfen Sie die korrekte Anzahl der Parameter über eine Assertion.
2. Starten Sie das Programm mit und ohne Verwendung von Assertions.