Programmierzettel

Enumerationen (Aufzählungen)

Dominikus Herzberg

Version 1.1, 2021-07-06

Lerninhalte

Enumerationen (Aufzählungen)

Syntax zur Klassendeklaration

```
ClassDeclaration: NormalClassDeclaration
                | EnumDeclaration
```

Eine Enumeration (Aufzählung) ist eine spezielle Klassendeklaration, die — anders als die "normale" Klasse — nicht mit `class`, sondern dem Schlüsselwort `enum` beginnt.

Mit der Enumeration ist ein Klassentyp nutzbar, zu dem es nur eine fixe Anzahl namentlich definierter Instanzen gibt, die Aufzählungs- bzw. Enumerationswerte heißen — manche nennen sie auch Enumerationskonstanten.

Beispiele für Aufzählungen sind die Tage einer Woche, die chemischen Elemente der Periodentafel, die verfügbaren Operatoren für arithmetische Operationen. Sehr nützlich sind Aufzählungstypen z.B. auch bei der Beschreibung der Zustände von Zustandsautomaten.

Syntax zur Deklaration einer Enumeration

```
EnumDeclaration:
    {ClassModifier} `enum` Identifier [Superinterfaces] EnumBody

EnumBody:
    `{` [EnumConstantList] [`,`] [EnumBodyDeclarations] `}`

EnumConstantList: EnumConstant {`,` EnumConstant}

EnumConstant:
    {EnumConstantModifier} Identifier [`(` [ArgumentList] `)`] [ClassBody]

EnumConstantModifier: Annotation

EnumBodyDeclarations:
    `;` {ClassBodyDeclaration}
```

Da Enumerationen vieles können, was auch Klassen können, konzentrieren wir uns hier auf die Besonderheiten, die mit der Angabe von Aufzählungswerten einhergehen, siehe `EnumConstant` in der Grammatik. Man kann Aufzählungswerte nicht nur auflisten, sondern mit einem Konstruktoraufruf samt Argumenten konfigurieren und einen Klassenrumpf deklarieren.

Eine Klasse kann eine Enumeration nicht mit `extends` erweitern!

Eine Aufzählung deklarieren: `enum`

```
enum Weekday {                        ①
  MON, TUE, WED, THU, FRI, SAT, SUN; ②
}
```

① Eine Aufzählung erbt implizit von der abstrakten Klasse `java.lang.Enum`, die u.a. die Interfaces `Constable` und `Comparable` implementiert (Methode `compareTo`)

② Jeder Aufzählungswert darf nur genau einmal vorkommen, seine Position in der Aufzählung entspricht seiner Ordinalzahl (Ordnungszahl, beginnend mit 0)

Aufzählungswerte werden per Konvention in GROSSBUCHSTABEN geschrieben

Das Wichtigste in Kürze

- Jedem Aufzählungswert wie z.B. `Weekday.MON` entspricht *genau eine* Instanz vom Aufzählungstyp `Weekday`; `Weekday.MON` liefert immer *dieselbe* Instanz zurück

Nützliche Methoden

```
jshell> Weekday.TUE.name() ①
$88 ==> "TUE"

jshell> Weekday.SUN.ordinal() ②
$89 ==> 6

jshell> Weekday.THU.compareTo(Weekday.MON) ③
$90 ==> 3

jshell> Weekday.values() ④
$91 ==> Weekday[7] { MON, TUE, WED, THU, FRI, SAT, SUN }

jshell> Enum.valueOf(Weekday.class,"MON") ⑤
$27 ==> MON
```

① Die Methode `name()` gibt den Aufzählungsidentifier als String zurück

② Die Methode `ordinal()` liefert die Ordinalzahl des Aufzählungswerts

③ Die Methode `compareTo` vergleicht Aufzählungswerte über die Differenz der Ordinalzahlen

④ Die statische Methode `values()` liefert ein Array der Aufzählungswerte, was praktisch zur Iteration über die Aufzählungswerte ist

⑤ Abruf des Aufzählungswerts über Enumerationstyp und Namensrepräsentation als Zeichenkette

Ein Konstruktor für Aufzählungswerte

Das folgende Beispiel listet zur Anschauung nur die ersten fünf chemischen Elemente des Periodensystems auf und nutzt die Ordinalzahl, um die Ordnungszahl des Elements abzubilden. Der Code mit den Instanzvariablen, dem Konstruktor und `toString` ist wie bei einer Klasse aufgebaut.

```
enum Element { // enumeration according to perodic table
    H("Wasserstoff", 1.0079),
    HE("Helium", 4.0026),
    LI("Lithium", 6.941),
    BE("Beryllium", 9.0122),
    B("Bor", 10.811);                                    ①

    final String name;                                   ②
    final double atomicWeight;

    private Element(String name, double atomicWeight) { ③
        this.name = name;
        this.atomicWeight = atomicWeight;
    }

    public String toString() {                           ④
        return this.name().substring(0,1).toUpperCase()
             + this.name().substring(1).toLowerCase();
    }
}
```

① Wenn "normaler" Code im Aufzählungstyp folgt, muss ein Semikolon das Ende der Aufzählungswerte abschließen

② Die Instanzvariablen eines Aufzählungswerts sollten unveränderlich sein und damit `final` gesetzt werden

③ Der Konstruktor *muss* `private` gesetzt sein, da dies kein "normaler", von außen verfügbarer Konstruktor ist; er wird einmal mit der Erzeugung des Aufzählungswerts aufgerufen

④ Ein Beispiel, wie man die Repräsentation eines Aufzählungswerts per `toString` verändern kann

Das Wichtigste in Kürze

- Ein Aufzählungswert kann einen optionalen Konstruktoraufruf haben
- Ein Aufzählungstyp ist im Prinzip eine Klasse und kann Member und Methoden haben; deshalb ist ein Konstruktor hinzufügbar
- Das Beispiel zeigt, wie Aufzählungswerte einen Konstruktor nutzen, um Informationen zum Aufzählungswert zu speichern.

Aufzählungen und `switch`, `for`, `stream()`

Die `switch`-Anweisung arbeitet mit Enumerationen zusammen und wird gerne dafür verwendet. Hier ein einfaches Beispiel:

```
boolean isWorkday(Weekday day) {
    switch(day) {
        case SAT:
        case SUN: return false;
        default: return true;
    }
}
```

Hinweis

- Im `case` wird jeweils nur der Aufzählungswert verwendet, also nur z.B. `SAT` statt `Weekday.SAT`

Zur Iteration über die Aufzählungswerte ist die sogenannte foreach-Variante der `for`-Schleife eine passende Wahl.

Ausgabe aller Elemente mit `for` (foreach)

```
jshell> for(Element e : Element.values())
   ...> System.out.printf("Atomgewicht von %s (%s): %f\n",e.name,e,e.atomicWeight);
Atomgewicht von Wasserstoff (H): 1,007900
Atomgewicht von Helium (He): 4,002600
Atomgewicht von Lithium (Li): 6,941000
Atomgewicht von Beryllium (Be): 9,012200
Atomgewicht von Bor (B): 10,811000
```

Verwendet man Streams, sind die Verarbeitungsmöglichkeiten sehr vielfältig; so sieht moderner Java-Code aus:

Gesamtes Atomargewicht mit `stream()`

```
jshell> Arrays.stream(Element.values()).
   ...> mapToDouble(e -> e.atomicWeight).
   ...> sum()
$20 ==> 31.7747
```

In der JShell muss der Punkt (`.`) ans Ende der Zeile gesetzt werden, sonst weiß die JShell nicht, dass nach dem Zeilumbruch weiterer Code folgt. In Code, der nicht von der JShell ausgeführt wird, wird der Punkt mit in die nächste Zeile übernommen.

Anonyme Klassen für Aufzählungswerte

Das Beispiel ergänzt jeden Operator um eine ihm eigene Implementierung einer Evaluierungsmethode mit Hilfe einer anonymen Klasse.

```
enum Operator {
    ADD { int eval(int x, int y) { return x + y; }},
    SUB { int eval(int x, int y) { return x - y; }},
    MUL { int eval(int x, int y) { return x * y; }},
    MOD { int eval(int x, int y) { return x % y; }},
    DIV { int eval(int x, int y) { return x / y; }};
    abstract int eval(int x, int y);                ①
}
```

① Ohne diese abstrakte Methode sind die `eval`-Methoden der Aufzählungswerte nicht von "außen" aufrufbar.

Das Wichtigste in Kürze

- Ein Aufzählungswert kann einen optionalen Klassenrumpf haben
- Dieser Klassenrumpf deklariert eine anonyme Klasse, die den Aufzählungstyp erweitert; es gelten die Regeln für anonyme Klassen
- Die in anonymen Klassen deklarierten Instanzmethoden sind von "außen" nur aufrufbar, wenn sie Methoden des Aufzählungstyps überschreiben

Interaktionsbeispiel

```
jshell> Operator.ADD.eval(3,7)
$128 ==> 10

jshell> Operator.SUB.eval(3,7)
$129 ==> -4

jshell> Operator.MUL.eval(3,7)
$130 ==> 21
```

Zwei Klassen: `EnumSet` und `EnumMap`

Für die Arbeit mit Enumeration gibt es zwei interessante Klassen: `EnumSet` und `EnumMap`. Zwei kurze Beispiele zu ihrer Verwendung:

Mit `EnumSet` ist ein Ausschnitt der Aufzählungswerte wählbar; interessant für Iterationen

```
jshell> EnumSet.range(Element.HE,Element.BE)
$24 ==> [He, Li, Be]
```

Eine Abbildung von Aufzählungswerten zu einem anderen Typ mittels `EnumMap` kann in vielerlei Hinsicht interessant sein: man kann Informationen mit den Werten assoziieren (im Beispiel einen englischen Bezeichner für den Operatorwert); stehen die Aufzählungswerte für Zustände eines Zustandsautomaten, können die Zustände mit möglichen Transitionen (Zustandsübergänge) assoziiert werden usw.

Erzeugung einer `EnumMap`

```
EnumMap<Operator,String> docEn = new EnumMap<>(Operator.class);
docEn.put(Operator.ADD, "Addition");
docEn.put(Operator.SUB, "Subtraction");
```

Beispielabruf eines assoziierten Wertes

```
jshell> docEn.get(Operator.ADD);
$63 ==> "Addition"
```