Datenklassen

Sven Eric Panitz

26. Oktober 2022

Inhaltsverzeichnis

1 Daten strukturieren

Um mit Daten strukturiert zu arbeiten, muss man die Möglichkeit haben, Einzeldaten, die zusammen inhaltlich in einem Zusammenhang stehen, zu kombinieren. Hierfür kennt Java Objekte.

1.1 Datenklassen

Möchte man Daten in Java zusammenfassen, so bieten sich Datenklassen an.

Die Definition einer Datenklasse beginnt mit dem Schlüsselwort `record`, sodass man auch itunter von Record-Klassen spricht.

Zum Beispiel für eine Zeichenkette, in der ein Ort angegeben wird, und eine Zahl für eine Temperaturmessung, lässt sich die Datenklasse `Wetter` definieren.

```
jshell> record Wetter(String ort, int temperatur){}
|  created record Wetter
```

Damit gibt es von nun an einen neuen Typ namens `Wetter`. Objekte von diesem Typ haben alle eine Ortsangabe und eine Temperaturangabe.

Es sieht fast wie eine Funktionsdefinition aus. Vor allen Dingen gibt es wie bei Funktionen Argumente. In diesem Fall zwei Argumente: `ort` und `temperatur`. Diese Argumente bezeichnen wir als Felder der Datenklasse.

Datenklassen mit mehreren Feldern bezeichnet man in der theoretischen Informatik als Produkttypen. Wenn wir `String` als die Menge alle Zeichenketten und `int` als die Menge aller 32-Bit Zahlen betrachten, dann ist der Typ `Wetter` das karthesische Produkt dieser beiden Mengen, nämlich die Menge aller Paare, die aus Elementen der beiden Mengen für die Felder gebildet werden können.

1.2 Objekte erzeugen

Um für eine bestimmte Datenklasse für Einzelwerte ein Objekt zu erzeugen, kennt Java das reservierte Wort `new`. Nach diesem wird der Name der Klasse geschrieben, für die ein Objekt erzeugt werden soll. Dann folgen die Einzelwerte als Argumente.

So lassen sich zwei Objekte der Datenklasse `Wetter` wie folgt erzeugen.

```
jshell> var w1 = new Wetter("Frankfurt", 20)
w1 ==> Wetter[ort=Frankfurt, temperatur=20]

jshell> var w2 = new Wetter("Offenbach", 18)
w2 ==> Wetter[ort=Offenbach, temperatur=18]
```

Man spricht davon, dass der Konstruktor der Datenklassen zum Erzeugen eines Objekts aufgerufen wird.

Der Aufruf eines Konstruktors ist ein Ausdruck, der ein neues Objekt der entsprechenden Klasse erzeugt und diese Klasse als seinen Typ hat.

Davon kann man sich in der JShell auch noch einmal von überzeugen, indem man sich die beiden Variablen anzeigen lässt:

```
jshell> /vars w1 w2
|    Wetter w1 = Wetter[ort=Frankfurt, temperatur=20]
|    Wetter w2 = Wetter[ort=Offenbach, temperatur=18]
```

1.3 Zugriff auf die Felder eines Datenobjekts

Es geht im Prinzip bei strukturierten Daten zunächst einmal darum, Einzeldaten zu größeren Daten zusammenzufassen, um dann aus den zusammengefassten Objekten wieder die Einzeldaten zu selektieren.

Für einen Ausdruck, der von Typ einer Datenklasse ist, können wir mit einem Punkt und anschließend den Feldnamen mit leeren Arguementklammern, wieder die Daten, die im Konstuktor für ein Feld übergeben wurden, selektieren.

Für die Variable `w1` vom Typ `Wetter` können wir mit `.temperatur()` den Wert der dort gespeicherten Temperatur erfragen:

```
w1.temperatur()
```

```
$73 ==> 20
```

Und mit `w1.ort()` erhalten wir wieder die bei der Objekterzeugung übergebene Zeichenkette für den Ort.

```
w1.ort()
```

```
$74 ==> "Frankfurt"
```

Verschiedene Objekte derselben Datenklasse haben auch verschiedene eigene Werte. Das Datenobjekt `w2` wurde mit einem anderen Temperaturwert erzeugt, der sich für dieses Objekt auch wieder selektieren lässt.

```
w2.temperatur()
```

```
$75 ==> 18
```

`temperatur()` und `ort()` werden auch als Selektorfunktionen der Datenklasse bezeichnet.

1.4 Funktionen für strukturierte Daten

Oft will man Funktionen definieren, die mit den Daten eines Datenobjekts rechnen und neue Daten erzeugen wollen.

Hierzu kann man Funktionen schreiben, die das Datenobjekt einer Datenklasse als Argument erhalten und dann mit Hilfe der Selektorfunktionen zu neuen Daten verrechnen.

Ein typisches Beispiel ist, wenn man für die Datenklasse `Wetter` die Temperatur in Fahrenheit und nicht in Grad Celsius berechnen möchte. Dafür kann man sich eine Funktion schreiben, die ein Argument vom Typ Wetter erhält.

```
double temperaturInFahrenheit(Wetter w){
  return w.temperatur() * 1.8 + 32;
}
```

Diese Funktion kann jetzt wie jede anderen Funktion aufgerufen werden, wenn das Argument vom korrekten Typ der Funktion ist.

```
temperaturInFahrenheit(w1)
```

```
$37 ==> 68.0
```

1.5 Funktionen als Methoden der Datenklasse

Eine der grundlegensten Ideen der Objektorientierung ist, die Funktionen, die mit den Objekten von strukturierten Daten arbeiten, in die Klassen zu schreiben, die die Struktur der Objekte definieren. In der objektorierten Umsetzung würde man also nicht die obige Funktion `temperaturInFahrenheit` mit einem Argument vom Typ `Wetter` definieren, sondern diese Funktion innerhalb der Datenklasse schreiben:

```
record Wetter(String ort, int temperatur){
  double temperaturInFahrenheit(){
    return temperatur() * 1.8 + 32;
  }
}
```

Jetzt erkennt man auch, warum am Ende der Definition der Datenklasse ein Paar geschweifter Klammern folgte. Innerhalb dieses Klammernpaares lassen sich Funktionen definieren, die mit den Daten der Datenklasse arbeiten.

Die Funktionen einer Klasse werden als Methoden bezeichnet. Um zu betonen, dass es sich um Methoden handelt, die auf den Daten eines konkreten Objekts arbeiten, werden sie genauer als Objektmethoden bezeichnet.

Innerhalb der Methoden einer Klasse können die Felder der Klasse verwendet werden, ohne dass man davor das Objekt schreiben muss, für das man dieses Feld selektieren soll.

Das liegt daran, dass Methoden immer für ein bestimmtes Objekt aufgerufen werden. Statt wie eben, der Funktion das Wetterobjekt als Argument zu übergeben, wird jetzt das Objekt dem Methodenaufruf vorangestellt. Die Methode selbst hat aber keine Argumente.

```
w1.temperaturInFahrenheit()
```

```
$4 ==> 68.0
```

Damit ist bei jedem Methodenaufruf klar, wie die einzelnen Werte innerhalb der Methode sind. Die dort selektierte Temperatur ist die vom Objekt in der Variablen `w1`.

Damit werden solche Methoden, die wir definieren können, genauso aufgerufen, wie die Selektorfunktionen.

```
w1.temperatur()
```

```
$5 ==> 20
```