Modellierung und Programmierung 1

Übung 4

Stefan Preußner

23. / 24. November 2020

Die `main`-Methode

- In Java muss es mindestens eine Methode mit der folgenden Signatur geben:

  ```
  public static void main(String[] args)
  ```

  (die Variable `args` kann auch anders bezeichnet werden)
- Die `main`-Methode dient als Einstiegspunkt bei der Ausführung des Programms (die eigentliche Programmausführung besteht im Kern darin, die `main`-Methode abzuarbeiten)
- Mehrere Klassen können eine solche `main`-Methode besitzen. Bei der Ausführung des Programms muss dann die `main`-Methode angegeben werden, welche abgearbeitet werden soll
  - Die Klasse, welche diese `main`-Methode enthält, ist die Hauptklasse des Programms

Statische und nicht-statische Methoden

In Java gibt es statische und nicht-statische Methoden:

- **Nicht-Statische** Methoden
  - können immer nur für konkrete Objekte aufgerufen werden, der Aufruf erfolgt mit der Syntax
    `Objektname.Methodenname`
  - stellen über das Schlüsselwort `this` eine Referenz auf das Objekt, für das eine Methode aufgerufen wurde, zur Verfügung
  - haben Zugriff auf statische und nicht-statische Attribute und Methoden der Klasse, zu der sie gehören

Statische und nicht-statische Methoden

- **Statische** Methoden
  - werden durch das Schlüsselwort `static` gekennzeichnet: `public static void main(String[] args)`
  - können über den Befehl `Klassenname.Methodenname` an einer beliebigen Stelle im Code aufgerufen werden
    - Bspw. gibt es in der Klasse `Math` die Methode `public static int round(float x)`, damit kann an einer beliebigen Stelle im Code mittels `Math.round(zahl)` eine `float`-Variable namens `zahl` gerundet werden

  - können von der Klasse, zu der sie gehören, nur statische Attribute und andere statische Methoden verwenden

Scope / Geltungsbereich von Variablen

- Jede Variable hat einen Geltungsbereich (*scope*), bspw.
  - sind lokale Variablen nur innerhalb eines Anweisungsblocks,
  - Parameter nur innerhalb einer Methode und
  - Instanzvariablen nur innerhalb einer Klasse gültig
- Geltungsbereiche können sich überlagern (variable shadowing), bspw. wenn der Parameter einer Funktion und das Attribut einer Klasse den gleichen Namen haben
  - in diesem Fall kann mit `this` explizit auf das Klassenattribut zugegriffen werden

super

- Das Schlüsselwort `super` referenziert in Java die direkte Oberklasse und hat zwei Funktionen:
- Mit `super()` wird der Konstruktor der Oberklasse aufgerufen
  - `super()` können auch Argumente übergeben, dann wird der entsprechend parametrisierte Konstruktor der Oberklasse aufgerufen
  - `super()` **muss immer die erste Anweisung in einem Konstruktor sein!**
  - Wenn kein Aufruf von super() angegeben ist, dann wird dieser automatisch eingefügt - d.h. es erfolgt immer ein Aufruf des Konstruktors der Oberklasse

super

- Mit `super` kann auf Methoden sowie Instanz- und Klassenvariablen der Oberklasse zugegriffen werden
  - Bsp.: super.toString() ruft die toString()-Methode der Oberklasse auf
  - Diese Funktionalität ist insbesondere bei überschriebenen Methoden sinnvoll
  - Der Zugriff auf Variablen ist nur insoweit möglich, wie deren Sichtbarkeit dies zulässt

instanceof

- Mit dem Schlüsselwort `instanceof` kann die Zugehörigkeit eines Objekts zu einer Klasse geprüft werden
- Syntax: `Objekt instanceof Klassenname`
- Es wird dann `true` zurückgegeben, wenn das Objekt zu einer Klasse gehört, die entweder der angegebenen Klasse entspricht oder eine ihrer Unterklassen ist
  - Bsp.: `Episode` erbt von `Film`. `ep` sei eine Instanz der Klasse `Episode`, `fi` sei eine Instanz der Klasse `Film`.
    - `ep instanceof Episode` → `true`
    - `ep instanceof Film` → `true`
    - `fi instanceof Episode` → `false`
    - `fi instanceof Film` → `true`

Typenkonvertierung / type casting

- Der Wert einer Variablen kann, unter bestimmten Bedingungen, einer Variablen mit einem anderen Datentyp zugewiesen werden
- Hierzu wird der Zieldatentyp in Klammern vor den umzuwandelnden Wert gesetzt
- Beispiel:

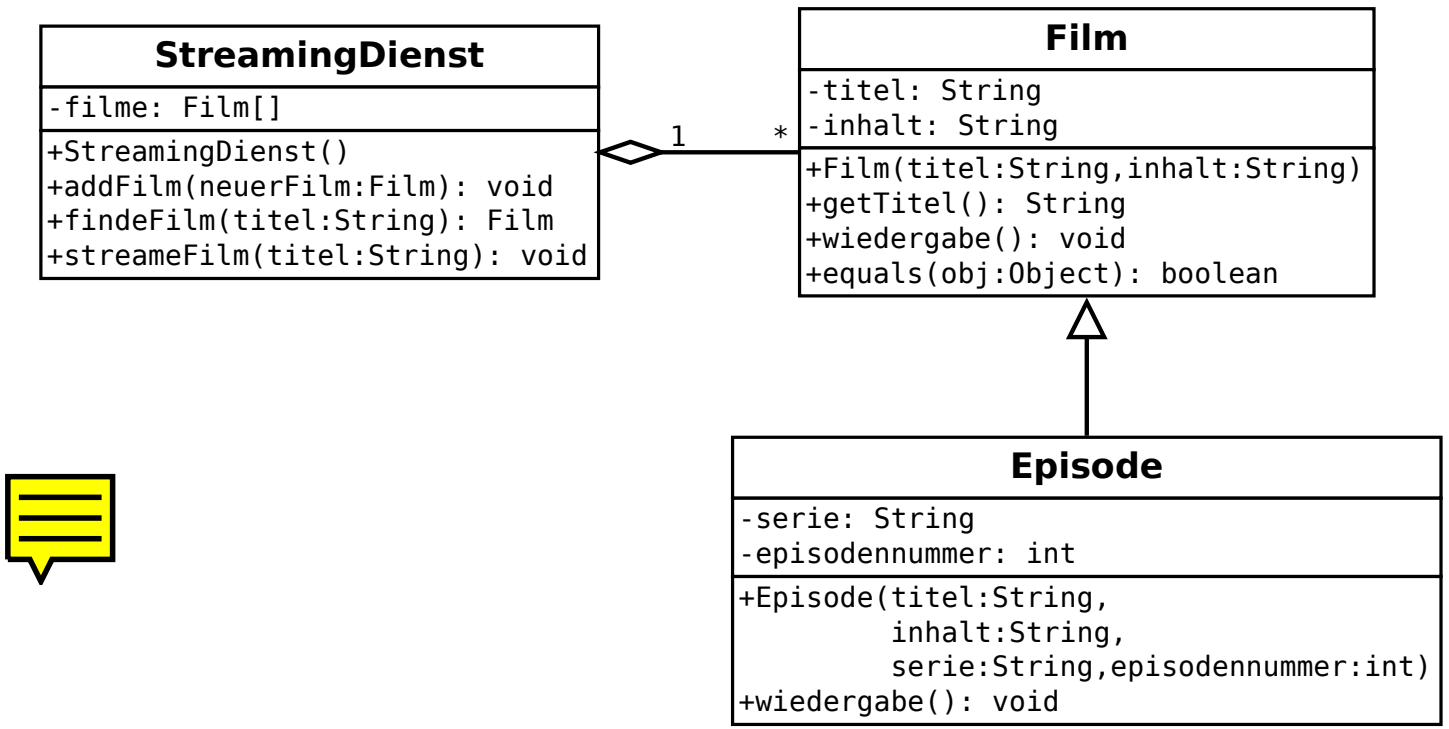
```
double x = 3.5;
int y = (int)x; // y hat den Wert 3
```

Typenkonvertierung / type casting

- Umgewandelt werden können u.a.:
  - Primitive Zahlendatentypen untereinander (`char` ist auch eine Zahl!)
  - Objekte einer Klasse X in Objekte einer Oberklasse von X
    - Wichtig: Die Information, zu welcher Klasse ein Objekt ursprünglich gehörte, wird zusammen mit den Instanzvariablen bei einer Typenkonvertierung weiterhin gespeichert
- Nicht umgewandelt werden können u.a.:
  - `boolean` in primitive Zahlendatentypen und umgekehrt
  - Objekte einer Klasse X in Objekte einer Unterklasse von X
    - Ausnahme: das Objekt gehört ursprünglich der Unterklasse an

Übung - Programmierung

Implementiert werden soll folgendes UML-Klassendiagramm:



Klasse StreamingDienst

- `filme` speichert alle Filme des Streamingdienstes
- Der Konstruktor soll ein neues `Film`-Array der Länge 100 erzeugen
- `addFilm` soll den übergebenen Film an der ersten freien Stelle in `filme` speichern
- `findeFilm` soll den Film mit dem übergebenen Titel zurückgeben. Existiert kein solcher Film, dann soll `null` zurückgegeben werden
- `streameFilm` soll den Film mit dem übergebenen Titel wiedergeben. Existiert kein solcher Film, dann soll eine Fehlermeldung ausgegeben werden

Klasse Film

- `titel` speichert den Titel, `inhalt` den Inhalt eines Films
- `wiedergabe` soll erst den Titel und dann den Inhalt ausgeben
- `equals` soll dann `true` zurückgeben, wenn das Vergleichsobjekt ein `Film` mit dem gleichen Titel ist

Klasse Episode

- `serie` speichert den Namen der Serie, zu der die Episode gehört
- `episodennummer` speichert die Nummer der Episode innerhalb der Serie
- `wiedergabe` soll vor der Ausgabe des Titels und des Inhalts (wie in der Klasse `Film`) die Serie und die Episodennummer ausgeben

Klasse Main

- In der `main`-Methode soll ein neuer Streamingdienst erstellt werden
- Neue Filme und Episoden sollen erstellt und dem Streamingdienst hinzugefügt werden
- Anschließend soll ein beim Streamingdienst gespeicherter Film wiedergegeben werden