Karlsruher Institut für Technologie

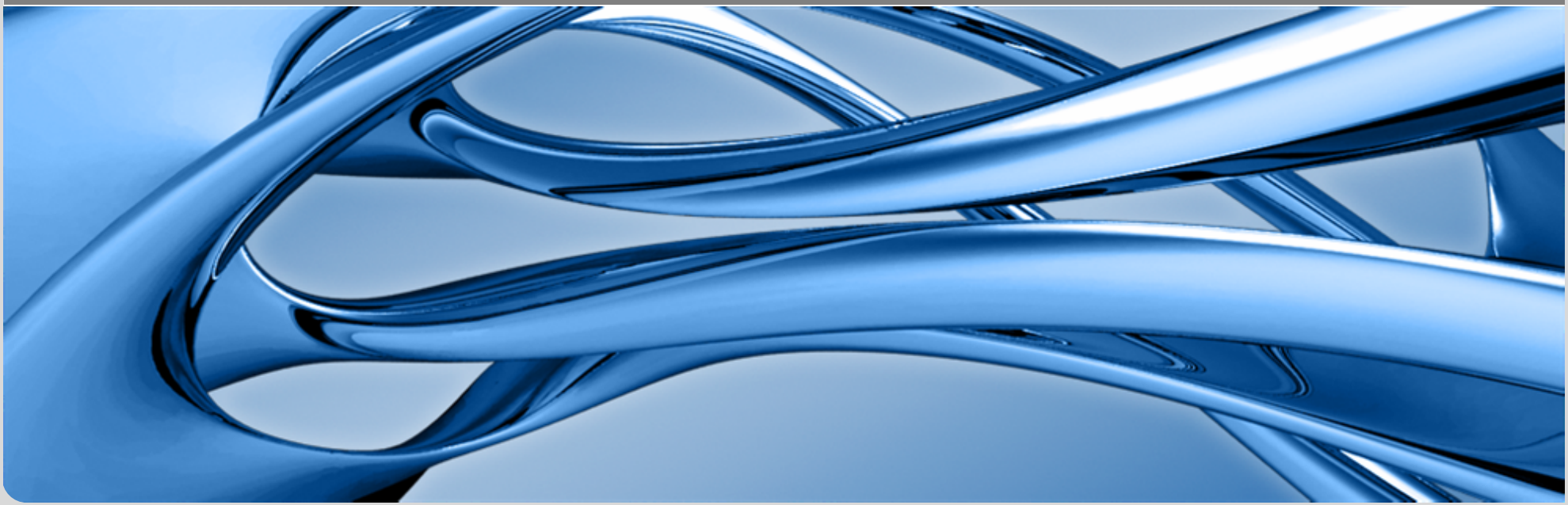
SDQ Software Design and Quality

Tutorium 7

Wiederholung und Vererbung

Christian Zielke | 11. Dezember 2018

CHAIR FOR SOFTWARE DESIGN AND QUALITY



Gliederung

1. Allgemeines
   - Übungsblatt 2
   - Übungsblatt 3
2. Wiederholung
   - enum
   - Getter und Setter
   - Übungsaufgabe

Gliederung

Übungsblatt 2

Plagiate

Nicht abschreiben!!!!!!!!!!!!!!!!!

Übungsblatt 2

Aufgabe A

- Ausgabe soll in main Methode erfolgen.
- Ausgabeformat beachten! (z.B. Semikolon am Ende)

Übungsblatt 2

Aufgabe B

- Singleton falsch verwendet
- Monat wird als 0-11 zurückgegeben => +1
- Ausgabeformat beachten! (Aktuelles Datum ...)

Übungsblatt 2

Aufgabe D

- Begründung für getter/setter.
- toString() begründen.
- Jede Klasse sollte toString() Methode besitzen.
- Genre soll als enum modelliert werden.
- Eine Klasse für Adresse und nicht eine für Studioadresse und eine für Schauspieleradresse.

Übungsblatt 3

Hinweise

- Optionale Checkstyle Regeln werden mit Punkten bewertet.
- Einführung der Terminal Klasse.

Wichtig

- Terminal Klasse nicht mit abgeben.
- Terminal Klasse nicht verändern.

enum

Der enum-Datentyp speichert **Aufzählungen**:

- enum <Name> {WERT1, WERT2, ...}
- Beispiel:

```
1   enum Weekday {
2     MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY
3   };
```

- ein enum kann/sollte in einer eigenen Datei angelegt sein
- es müssen alle Werte vorher bekannt sein
- Eine enum-Variable kann einen der aufgezählten Werte annehmen:

```
1   Weekday day = Weekday.THURSDAY;
```

enum

```
1   public enum Month {
2     JANUARY(31),
3     FEBRUARY(28),
4     MARCH(31),
5     APRIL(30);
6
7     private final int days;
8
9     private Month (int days) {
10      this.days = days;
11    }
12
13    public int getDays() {
14      return days;
15    }
16
17    @Override
18    public String toString() {
19      return this.name().toLowerCase();
20    }
21  }
```

enum

Dokumentation und Beispiele

`Klick`

Getter und Setter

```
1   public class MyNumber {
2     private int number;
3
4     public MyNumber(int number) {
5       this.number = number;
6     }
7
8     public int getNumber() {
9       return number;
10    }
11    public void setNumber(int number) {
12      this.number = number;
13    }
14  }
```

Übungsaufgabe

Besprechung Aufgabe D

Was sind sinnvolle getter und setter?

Grundlagen der Vererbung

- *”Mechanismus zur Implementierung von **Generalisierung** bzw. **Spezialisierung**”*
- Dabei gilt:
  - **Unterklasse:** speziellere Klasse (subclass)
  - **Oberklasse:** generellere Klasse (superclass)
- Beispiel: evolutionäre Klassifizierung
  - Krokodile sind Reptilien
    $\rightarrow$ **super**: Reptil; **sub**: Krokodil
  - Vögel sind Wirbeltiere
    $\rightarrow$ **super:** Wirbeltier; **sub:** Vogel
  - Reptilien sind Wirbeltiere
    $\rightarrow$ **super:** Wirbeltier; **sub:** Reptil
- **ist-ein** Beziehung
- Es kann immer nur von einer Klasse geerbt werden

Was wird vererbt?

- Vererbt werden:
  - Attribute
  - Methoden
  - Geschachtelte Klassen (z.B. Iterator vom letzten Mal)
  - **KEINE** Konstruktoren
- Die Unterklasse kann durch Funktionen erweitert werden
- Daraus folgt **Substituierbarkeit:**
  Unterklasse kann anstelle der Oberklasse eingesetzt werden
  $\rightarrow$ Wenn du dir zu Weihnachten ein Handy wünscht,
  kannst du nicht ausschließlich ein iPhone erwarten

Syntax

- Crocodile **erbt von** Reptile
  → `Crocodile` **extends** `Reptile`
- Reptile r = new Crocodile()
  → Erzeugt ein Objekt vom Typ `Crocodile` und speichert dieses in einer Variable vom Typ `Reptile`
- Crocodile c = new Reptile()
  → **NICHT MÖGLICH**

Überschreiben (Override)

- Eine geerbte Methode kann bei Bedarf **überschrieben** werden
- Voraussetzungen:
  - gleicher Name
  - gleiche Parameter (Anzahl und Typen)
  - Rückgabetyp ist Subtyp des Vorherigen oder gleich
- Signalisierung durch die Annotation `@Override`

Überschreiben (Override)

```
1   class Reptile {
2     public void bite() {
3       ...
4     }
5   }
6
7   class Crocodile extends Reptile {
8
9     @Override
10    public void bite() {
11      ...
12      rotate();
13    }
14  }
```

Dynamische Bindung

```
1   Crocodile c = new Crocodile();
2   Reptile r = new Reptile();
3   Reptile rep = c;
```

- Ruft man eine Methode auf einem Objekt auf, wird erst **zur Laufzeit** festgelegt, welche Methodendefinition verwendet wird
- `c.bite()` führt Code von Crocodile aus
- `r.bite()` führt Code von Reptile aus
- `rep.bite()` führt Code von **Crocodile** aus

Schlüsselwort super

- Mit `super` können Attribute und Methoden der Oberklasse aufgerufen werden (analog zu `this`)
- Bei Methoden wird hier einmalig **nicht** dynamisch gebunden

Konstruktoren

- Konstruktoren werden zwar nicht vererbt, der Konstruktor der Oberklasse kann aber per `super(...)` aufgerufen werden
- Dies geschieht sogar immer direkt am **Anfang** von jedem Konstruktor
- Wird nicht explizit ein bestimmter Super-Konstruktor aufgerufen, wird implizit `super()` ausgeführt
- Problem: Existiert in der Oberklasse kein Default-Konstruktor, führt dies zu Fehlern
  $\rightarrow$ In diesem Fall müssen in allen Subklassen Konstruktoren definiert sein

Operator instanceof

- Der `instanceof` prüft, ob ein Objekt den gewünschten Typ hat
- Syntax: `Objekt instanceof Klasse`
- Beispiel:

```
1     static boolean isCrocodile(Reptile r) {
2       if (r instanceof Crocodile) {
3         return true;
4       } else {
5         return false;
6       }
7     }
```

- Sparsam und nur wenn unbedingt nötig (z.B. bei Casts) verwenden!

Die Klasse Object

- `Object` ist die Oberklasse aller anderen Klassen (Wurzel des Vererbungsbaumes)
- Methoden aus `Object` müssen bei Bedarf **überschrieben** werden
- Wichtige Methoden sind `equals()`, `toString()` und `clone()`

Sichtbarkeit

- Zusätzlichen Modifikator `protected`
- Sichtbarkeit zwischen public und package
- Zusätzliche Sichtbarkeit in Unterklassen

Vererbbarkeit verhindern: final

- Möchte man eine Klasse nicht vererbbar machen, so setzt man diese **`final`**
- Möchte man Überschreiben einer Methode verhindern, so setzt man diese **`final`**

Abstrakte Klassen: abstract

- Möchte man Vererbung verwenden, die Oberklasse selbst macht aber keinen Sinn (z.B. ein Reptil an sich existiert nicht) kann man diese Klasse als **abstract** deklarieren
- Die Klasse kann dann nicht instantiiert werden, sondern nur vererbt
- Auch Methoden können als abstract deklariert werden, diese müssen in Unterklassen implementiert werden

Abstrakte Klassen: abstract

```
1   abstract class Reptile {
2     ...
3     abstract void bite();
4   }
5
6   public class Crocodile extends Reptile {
7     public void rotate() {
8       ...
9     }
10
11    @Override
12    public void bite() {
13      ...
14      rotate();
15    }
16  }
```

Übungsaufgabe

Aufgabe

Implementiere folgende Klasse: MotorizedVehicle (superclass), Car (subclass), Truck (subclass), ...

MotorizedVehicle

- abstract
- überlegt euch sinnvolle Attribute
- Methoden: boolean drive(double distance), *boolean refuel(double amount)*, ...

Diagramm

siehe Tafel