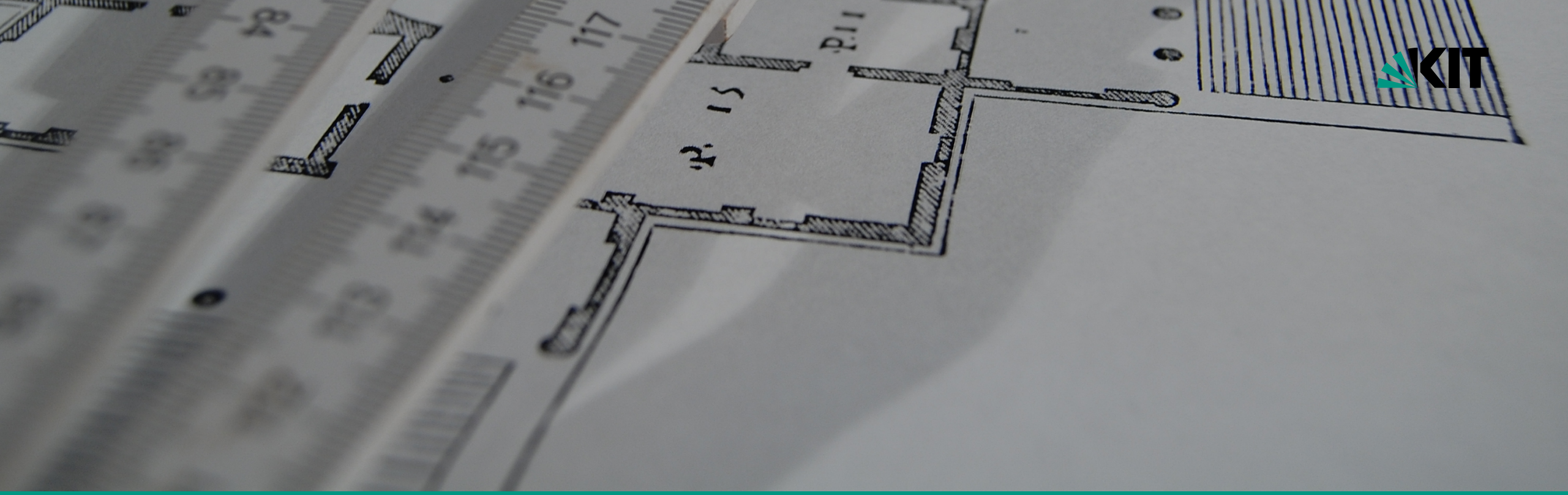

Vererbung

Programmieren Tutorium Nr.12

Aleksandr Zakharov | 16. Dezember 2025

Übungsblatt 2 - Besprechung

- Je Klasse eigene Datei
- Enum nicht nur anlegen, auch das passende Attribut
- Keine inline enums
- Objektorientierung! (Klassen als Datentyp nutzen, `...`)
- Datenkapselung (`private ...`)
- Wiederverwendung eigener Methoden (Vermeidung von Code-Duplizierung)
- Utility-Klassen: final class, privaten Konstruktor und die Methoden/Attribute static. Und **keine Instanzen** dessen!
- `if(var == true)` ⇔ `if(var)`
- `equals()` verwenden
- Sinnvolle Bezeichner & Kommentare
- Checkstyle

Übungsblatt 2 - Besprechung

- Falscher Schleifentyp (while statt for, for statt for-each verwendet)
- Nicht die schon gegebene Funktionalität reimplementieren!
- Mit `chars` direkt rechnen, nicht die ASCII-Tabelle-Kodierungen als integers verwenden!
- Auskommentierten Quelltext nicht abgeben
- Magic Numbers
- Vergleichsrichtung: `"text".equals(var)`
- Bitte kein `while(true)`
- Kompilieren, Ausführen, **TESTEN**
- Im Artemis nachschauen ob eswirklich kompiliert hat!

Weitere Anmerkungen

- Der Name einer Variablen hat nichts mit der Objektidentität zu tun:

```
FireTruck truck = new FireTruck();
FireTruck anotherTruck = truck;
// Beide Variablen zeigen auf den GLEICHEN Truck
truck == anotherTruck // true
```

- Eine Variable ist nicht notwendig, um ein Objekt zu erstellen

```
// Erzeugt einen FireTruck und ruft dessen Konstruktor auf
new FireTruck();
```

Immutable Objects

- Ein „Immutable Object“ ist ein Objekt bei dem der Zustand lediglich einmal am Anfang gesetzt wird und danach nie wieder verändert werden kann.
  Bsp.: alle Strings
- „Immutable Objects“ sind z.B. sehr hilfreich wenn die entsprechenden Objekte kopiert werden müssen, da dann lediglich die Referenz kopiert werden muss.
- Sie enthalten oftmals nur *Daten* $\Longrightarrow$ `equals` (`hashCode`) überschreiben lohnt sich wahrscheinlich
- Siehe Wikipedia - Immutable Object
- In Java ist es durch den `Record`-Klassentyp dargestellt (ihr könnt damit während den Ferien herumspielen)

Organisatorisches

Übungsblatt 3 Abgabe

10. Dezember 2025 12:00 Uhr - 18. Dezember 2025 6:00 Uhr

Warnung!

Warning - Warning - Warning

Dieses Tutorium beinhaltet heute teils (sehr) schw. . . herausfordernde Inhalte!

Was ist Vererbung? - Motivation

Biologie . . .

- Hund, Pudel, Katze, Vogel, Road Runner, Kojote
  Findet ihr Oberbegriffe?
- Findet ihr gemeinsame Verhaltensweisen oder Eigenschaften?
- Unterscheiden sich die Verhaltensweisen vielleicht immer etwas?

. . . und die Informatiker

- Findet ihr hier vielleicht Klassen (Mit Methoden!)?
- Haben diese Klassen Gemeinsamkeiten?

Was ist Vererbung? - Motivation

Und wo ist jetzt die Vererbung?

- Manche Dinge kann jedes Lebewesen. Bewegen zum Beispiel. Hat man nun also eine Liste von Lebewesen, kann man sie alle zum Bewegen bringen – ungeachtet ihrer Gattung oder Art
- Dieses Konzept (Gemeinsamkeiten im Verhalten oder Eigenschaften) liegt der Vererbung zu Grunde.
- ```
  Animal animal = new Dog();
  animal.moveTo(sierra);
  animal = new Bird();
  animal.moveTo(farSouth);
  ```
  ```

Vererbung

Wozu Vererbung?

Wir verwenden Vererbung für:

- Spezialisierungen (`moveTo` in einem Vogel macht nicht das gleiche wie in einem Hund!)
- Code-Wiederverwendung (`getAge` ist vielleicht überall gleich)
  - Weniger Code-Redundanz $\rightarrow$ erhöhte Wartbarkeit

Vererbung

Merke . . .

- Modelliert *ist-ein-Beziehungen*
  (ein Student ist ein Mensch, ein Mensch ist ein Säugetier, ein Säugetier ist ein Lebewesen)
- Kennzeichnung mittels „`extends`“
- Keine Mehrfachvererbung in Java (nur eine Oberklasse je Klasse)
- Jede Klasse erbt implizit von Object (equals, toString, hashcode, . . .)
  Dies ist wie das „Lebewesen“ in unserem Beispiel. Alles ist ein Lebewesen und in Java ist alles ein Objekt.
- „`final`“ verhindert Vererbung von der Klasse
- Grundsätzlich gilt das **Liskov'sche Substitutionsprinzip** Jede Instanz einer Subklasse sollte auch als Instanz einer Superklasse einsetzbar sein
  Beispiel: Wenn nur ein Objekt der Klasse `Animal` gefordert ist, sollte insbesondere auch ein Objekt der Klasse `Dog` in dem Kontext funktionieren, da ein `Dog` ja ein `Animal` ist.

Vererbung - Überschreiben

Gut zu wissen ...

- Angenommen, der Standard für `moveTo` für Lebewesen ist Laufen.
- Wie ändern wir `moveTo` in unserem Vogel, sodass er fliegt? Überschreiben!

```
class Bird {
    // Hilfs-Marker fuer den Compiler
    @Override
    // gleicher Name, Parameter und return type
    public void moveTo(Location location) {
        // Was es bei einem Aufruf machen soll
        flyTo(location);
    }
}
```

- Damit wird nun bei jedem Aufruf von `moveTo` `flyTo` aufgerufen

Vererbung - Überschreiben

Gut zu wissen . . .

- Will man Methodenverhalten von Oberklassen ändern, so kann man diese Methoden überschreiben
- Methodenname muss identisch zu dem in der Oberklasse sein
- Parameter müssen identisch zu denen in der Oberklasse sein
- Rückgabewert muss identisch zum Rückgabewert der Methode in der Oberklasse sein (gleich oder Subtyp)
- ’final’ Methoden können nicht überschrieben werden
- *@Override* stellt sicher, dass Überschreiben tatsächlich stattfindet (sonst Compilerfehler)
  - Überschriebene Methoden am besten IMMER mit *@Override* kennzeichnen

Vererbung - Überschreiben - Beispiele

```java
public class Mutter {
    public void anstrengend(int k) {
        System.out.println(k + "Stueck Kuchen sind genug!");
    }
}

public class Oma extends Mutter {
    @Override
    public void anstrengend(int j) {
        System.out.println("Nimm doch noch " + j + " Muffins.");
    }
}

public class Uroma extends Oma {
    @Override
    public void anstrengend(int i) {
        System.out.println("Nimm doch noch "+ i +" Stueck Kuchen.");
    }
}
```

Überladen

Gut zu wissen . . .

- Es kann mehrere Methoden mit gleichem Namen geben. Diese müssen sich aber in Art und/oder Anzahl ihrer Parameter unterscheiden
  - void doSomething(int a)
  - void doSomething(int a, int b)
  - void doSomething(int a, String b)
  - void doSomething(int a, int b, double c)
  - void doSomething(int a, long b)
- Dies gilt ebenso für Konstruktoren
- Dies hat **NICHTS** mit Vererbung zu tun!

Überladen - Beispiele

```java
public class Mutter {

  public void anstrengend(int k) {
    System.out.println(k + "Stueck Kuchen sind genug!");
  }

  public void anstrengend(String s) {
    System.out.println("So viel Kuchen tut dir nicht "+ s +"!");
  }

  public void anstrengend(double i) {
    System.out.println("Du kannst nur "+ i +" Stueck haben.");
  }

}
```

Überladen - Beispiele

Beispiele

- Was gibt `anstrengend(2)` aus?
- Was gibt `anstrengend("Bobby Tables")` aus?
- Was gibt `anstrengend(2.5)` aus?
- Was gibt `anstrengend(2.5f)` aus?
- Was gibt `anstrengend(2L)` aus?

Debugging

Was ist das?

Debugging ist der Prozess des Identifizierens und Illiminierens Von Fehlern (Bugs) im Code.

Wie geht das?

- Gute Frage...
- Ist immer sehr komplex und eigentlich das zeitintensivste am Entwicklungsprozess
- Zum Glück gibt es ein paar standartisierte Tools und Verfahren, die einem helfen können

Debugging

Tools & Verfahren

- Debugger in der IDE
- Benutzt den! Der kann sehr hilfreich sein!
- Wilde im code Verstreute print statements mit mehr oder weniger hilfreichen Ausgaben
  - Meistens unübersichtlicher als ein Debugger
  - Man vergisst gerne mal Ausgaben wieder raus zu nehmen
  - Kann für kleine Sachen trotzdem sehr praktisch sein
- Rubber ducky debugging
  - Mit einer anderen Dingen/Tieren/Menschen über den Code reden während man versucht den Fehler zu finden

Kurze Demo zum Debugger...

Aufgabe

Formen

In dieser Aufgabe sollen verschiedene geometrische Formen modelliert werden.
Jede Form soll Dabei durch ein Array von Punkten definiert sein, eine Farbe haben, seine Fläche berechnen können, um einen Vektor verschoben, um θ Grad um den Koordinatenursprung gedreht[1], und ausgegeben werden können.
Ale Formen sind 2-Dimensional. Konkret sollen Dreiecke, Quadrate, Rechtecke, Sechsecke und Kreise modelliert werden können.

[1] Dafür hilft es vermutlich zu wissen was eine Rotationsmatrix ist.