HTW D

Vorlesung Programmieren II

# OOP: Grundkonzepte und Entwurfsprinzipien

**Robert Ringel**
HTW Dresden

18.06.2025


© HTW Dresden

# Themenüberblick

Hochschule für Technik und Wirtschaft Dresden
University of Applied Sciences

Grundlagen der objektorientierten Programmierung und
die programmiersprachlichen Mittel von Java

Ziele und Vorteile von OOP

Entwurfsprinzipien zur Erreichung der Ziele

Quellen:

LAHRES, RAYMAN: Objektorientierte Programmierung 2., aktualisierte und erweiterte Auflage, Rheinwerk Computing, ISBN 978-3-8362-1401-8
https://openbook.rheinwerk-verlag.de/oop/oop_kapitel_03_001.htm#mja6c3f342e440956adade4ec377b02c1a
https://www.ionos.de/digitalguide/websites/web-entwicklung/solid-principles/
https://www.baeldung.com/solid-principles

Die nachfolgenden Codebeispiele wurden teilweise diesen Quellen entlehnt.

# Themenüberblick

Hochschule für Technik und Wirtschaft Dresden
University of Applied Sciences

**Ziele der Software-Entwicklung**
Hohe Produktqualität und akzeptable Kosten … durch:
- Modularität
- Wiederverwendbarkeit
- Erweiterbarkeit, Skalierbarkeit
- Flexibilität
- Schnittstellenbeschreibung
- Sicherheit

**Entwurfsprinzipien**
z. B. SOLID
- Single Responsibility
- Open-Closed-Principle
- Liskov-Substitution-Principle
- Interface-Segregation-Principle
- Dependency Inversion


**Problemlösung**

**Grundkonzepte der objektorientierten Programmierung**
- Abstraktion
- Datenkapselung
- Beziehungen
- Polymorphie
- Wiederverwendung durch gute Modellierung

**Programmiersprachliche Mittel in Java**
package, class, interface, abstract, extends, implements
Zugriffsmodifizierer: private, protected, public
allgem. Modifizierer: static, final, abstract

# Grundkonzepte der objektorientierten Programmierung

Hochschule für Technik und Wirtschaft Dresden
University of Applied Sciences

**Abstraktion**
Idee der Trennung von Konzept und konkreter Realisierung
… findet Ausdruck in Begriffen wie Klasse und Instanz, Schnittstelle und Implementierung oder auch in abstrakten Klassen zur Definition von Konzepten.

**Kapselung**
Klasse als eine Kapsel von Daten und definiertem Verhalten, wobei der Zugriff auf die Daten nur mit wohldefinierten Methoden möglich ist.
… Verminderung der Komplexität der Nutzung von Objekten über einfache öffentliche Schnittstellen (Methoden)

**Beziehungen**
Klassen und Objekte stehen in Beziehungen zueinander
- Generalisierung/Spezialisierung (Vererbung)
- Aggregation / Komposition (Teil-von-Beziehung)
- Verwendung oder Aufruf-Beziehung

**Polymorphie**
Als Mittel um spezifisches Verhalten einzelner Klassen unter generischen Begriffen (Methodennamen) abzubilden (Beispiele: bewegen, darstellen). Polymorphie steht häufig in Verbindung zur Spezialisierung.

**Wiederverwendung**
Abstraktion, Kapselung, Beziehungen und Polymorphie sollen durch eine gute Modellierung die Wiederverwendung von Klassen(-Bibliotheken) fördern. Dabei geht es weniger um spezifische Problemlösungen, sondern vielmehr und generische Lösungskonzepte.

# Grundkonzepte der objektorientierten Programmierung

Hochschule für Technik und Wirtschaft Dresden
University of Applied Sciences

**Abstraktion**
Idee der Trennung zwischen Konzept und konkreter Realisierung

*abstract*
*interface*

**Kapselung**
Klasse als eine Kapsel von Daten und definiertem Verhalten, wobei der Zugriff auf die Daten nur mit wohldefinierten Methoden möglich ist.

*public / private / protected*
*get / set*

**Beziehungen**
Klassen und Objekte stehen in Beziehungen zueinander

*extends / implements / packages*
*Instanzen von Klassen erzeugen und benutzen*
*static-Methoden nutzen*

**Polymorphie**
Als Mittel um spezifisches Verhalten einzelner Klassen unter generischen Begriffen (Methodennamen) abzubilden.

*Überschreiben von Methoden (z.B. toString)*
*@Override*

**Wiederverwendung**
Abstraktion, Kapselung, Beziehungen und Polymorphie sollen durch eine gute Modellierung die Wiederverwendung von Klassen(-Bibliotheken) fördern. *Modellierungsprinzipien*
*Entwurfsmuster*

# SOLID als Entwurfsprinzip

Hochschule für Technik und
Wirtschaft Dresden
University of Applied Sciences

Ursprung: Robert C. Martin („Uncle Bob“), „Design Principles and Design Patterns“ (2000)

SOLID als Gemeinsachaftswerk von:

Robert C. Martin, Bertrand Meyer und Barbara Liskov sowie Michael Feathers

**Basiert auf den fünf Prinzipien**
- Single Responsibility
- Open-Closed-Principle
- Liskov-Substitution-Principle
- Interface-Segregation-Principle
- Dependency Inversion

# SOLID als Entwurfsprinzip

Hochschule für Technik und
Wirtschaft Dresden
University of Applied Sciences

**Single Responsibility**
Prinzip der einzigen Verantwortung

Um Komplexität, Abhänigkeiten und Wechselwirkungen zu reduzieren soll jede Klasse nur eine einzige klar definierte Verantwortlichkeit realisieren.

Ziel: Vermeidung von fachlichen Überschneidungen!

```java
public class Verse {

	private String text;

	public Verse(String text) {
		this.text = text;
	}

	public String getText() {
		return this.text;
	}

	public void printFormattedText() {
		System.out.println(">>> " + this.text + " <<<");
	}
}
```

Negativbeispiel – warum?

# SOLID als Entwurfsprinzip

Hochschule für Technik und Wirtschaft Dresden
University of Applied Sciences

**Open-Closed-Principle**
Offen für Erweiterungen –
geschlossen für Änderungen

Klassen sollen leicht erweiterbar sein, um unnötige “Code-Adaptionen”, die zu Nebeneffekten, Komplikationen und Fehlern führen, zu vermeiden.

```
public class Verse {

     private String text;
     private String info;

     public Verse(String text, String info) {
          this.text = text;
          this.info = info;
     }

     public String getText() {
          return this.text;
     }

     public String getInfo() {
          return this.info;
     }

     public String getAuthor() {
          if (info.indexOf("author:") >=0 )
               return info.substring(info.indexOf("author:"));
          return "";
     }
}
```

Negativbeispiel – warum?

# SOLID als Entwurfsprinzip

Hochschule für Technik und Wirtschaft Dresden
University of Applied Sciences

## Liskov-Substitution-Principle

Eine durch Spezialisierung abgeleitete Klasse muss alle Methoden der übergeordneten Klassen fachlich sinnvoll ausführen können.

Ziel: Erhöhung der Sicherheit und Stabilität beim Austausch von Implementierungen in Subklassen oder Superklassen.

```
public class Rectangle {
     private float a;
     private float b;

     public Rectangle(float a, float b) {
          this.a = a; this.b = b;
     }

     public void scale_a(float f) {
          this.a = f * this.a;
     }

     public void scale_b(float f) {
          this.b = f * this.b;
     }
}

public class Square extends Rectangle {

     public Square(float a) {
          super(a, a);
     }

     public void scale(float f) {
          super.scale_a(f);
          super.scale_b(f);
     }
}
```

Negativbeispiel – warum?

# SOLID als Entwurfsprinzip

HTW D
Hochschule für Technik und Wirtschaft Dresden
University of Applied Sciences

**Interface-Segregation-Principle**
Interfaces sollten funktionsspezifisch und treffend zugeschnitten sein, um keine unnötigen Abhängigkeiten zu schaffen.

Durch Nutzung von Interfaces soll kein Zwang entstehen, unnötige Funktionalität zu implementieren.
Also große Interfaces in kleinere Interfaces zerlegen.

```java
public interface BearKeeper {
    void washTheBear();
    void feedTheBear();
    void petTheBear();
}

public class BearCarer implements BearKeeper {

    public void washTheBear() {
        //I think we missed a spot...
    }

    public void feedTheBear() {
        //Tuna Tuesdays...
    }

    public void petTheBear() {
        //Good luck with that!
    }

}
```

Negativbeispiel – warum?

# SOLID als Entwurfsprinzip

HTW D
Hochschule für Technik und Wirtschaft Dresden
University of Applied Sciences

## Dependency Inversion

Übergeordnete Klassen sollen nicht von untergeordneten Klassen abhängen. Abstraktionen (Schnittstellen) sind zu nutzen, um die Abhängigkeiten aufzulösen. Abstraktionen sollen nicht von Details abhängen.

Ziel: Abhängigkeiten mit Hilfe abstrakter, zwischengelagerter Schnittstellen auflösen.

```java
public class Lamp {
   private boolean lights = false;
   public void switch_on() {
      lights = true;
   }
   public void switch_off() {
      lights = false;
   }
}

public class Switch {
   private Lamp lamp;
   private boolean pressed;
   public Switch(Lamp lamp) {
      lamp = lamp;
   }
   public void press() {
      pressed = !pressed;
      if(pressed) {
         lamp.switch_on();
      } else {
         lamp.switch_off();
      }
   }
}
```

```java
public interface Device{
   public void switch_on();
   public void switch_off();
}

public class Lamp implements Device {
   private boolean lights = false;
   public void switch_on() {
      lights = true;
   }
   public void switch_off() {
      lights = false;
   }
}

public class Switch {
   private Device lamp;
   private boolean pressed;
   public Switch(Device lamp) {
      lamp = lamp;
   }
   public void press() {
      pressed = !pressed;
      if(pressed) {
         lamp.switch_on();
      } else {
         lamp.switch_off();
      }
   }
}
```

# SOLID als Entwurfsprinzip

Hochschule für Technik und
Wirtschaft Dresden
University of Applied Sciences

## Wenn der Code nicht SOLID ist ...

steigt der Wartungsaufwand und ggf. sinkt die Funktionssicherheit.

Code smells … historisch gewachsene Schwachstellen aus unsauberem Code; Funktionsfehler, inkompatible Programme/Klassen.

Code rots … bei ausbleibender Reparatur, Refactoring, Review kann der Code seine gesamte Funktion verlieren.

Sicherheitslücken … sind zeitnah zu schließen bzw. gezielt zu suchen, um sie schließen zu können.

**Was tun um soliden Code zu schreiben?**

- 4-Augen-Prinzip … Review durch erfahrene Entwickler/Programmierer
- nicht alles selber neu erfinden … Entwurfsmuster nutzen

# Andere Entwurfsprinzipien

Hochschule für Technik und Wirtschaft Dresden
University of Applied Sciences

**DRY-Prinzip** (Don’t repeat yourself)
"Every piece of knowledge must have a single, unambiguous, authoritative representation within a system".

Jede Funktionalität und jedes Datenelement sollte an einer einzigen, eindeutigen Stelle implementiert sein.
Erhöht das Verständnis und die Klarheit, reduziert Wartungsaufwand

Ursprung: The Pragmatic Programmer, 20th Anniversary Edition, David Thomas and Andrew Hunt, 2019, Addison Wesley, ISBN 978-0135957059.

/* Gegenteil ist WET: write every time, we enjoy typing, waste everyone's time */

**KISS-Prinzip** (Keep it simple, stupid)
Entwurfsziel für einen möglichst einfach konstruierten Code

# Zusammenfassung

Hochschule für Technik und Wirtschaft Dresden
University of Applied Sciences

Es gibt Programmierprinzipien, die einen sauberen, sicheren, wartbaren und damit wirtschaftlichen Code ermöglichen.

Das Programmieren und insbesondere der Programmentwurf sind verantwortungsvoll, gründlich und abwägend durchzuführen, wenn es wirklich gut werden soll.

Die erfolgreiche Anwendung der Programmierprinzipien bedarf viel praktischer Programmiererfahrung.

> ”Die Zeit, derer es bedarf, damit sich aus einem Anfänger ein Experte entwickelt, wird von Winslow mit ca. zehn Jahren angegeben. Im Rahmen eines vierjährigen Studiums ist es danach nicht möglich, den Übergang vom Anfänger zum Experten zu bewältigen. Winslow (1996, S. 21) gibt für den Studienabschluss das Qualifikationsniveau des kompetenten Programmierers als realistisches Ziel an.
> Winslow, L. E. (1996). Programming Pedagogy - a Psychological Overview. SIGCSE Bull., 28 (3), 17–22.”
>
> Ringel, R. (2024). Entwicklung und Evaluierung eines Rahmenkonzepts zum Programmierenlernen an Hochschulen
> https://nbn-resolving.org/urn:nbn:de:bsz:520-qucosa2-933983

Was tun?

Programmieren, programmieren, programmieren aber möglichst SOLID

… zum Beispiel die APL reviewen und überarbeiten

oder Praktikum 8 von Grund auf SOLID entwickeln über die Sommerpause

… damit das Programmieren nicht verlernt wird.

HTW D Hochschule für Technik und Wirtschaft Dresden University of Applied Sciences

# Vielen Dank!

**Robert Ringel**
Fakultät Informatik/Mathematik, HTW Dresden
https://www.htw-dresden.de/robertringel

T +49 351 462 2797
robert.ringel@htw-dresden.de