**http://ovm-kassel.de	Information**	
Information IT-AE-UML-INFO-3.4 Objektorientierte Programmierung (OOP) mit Java	
Code	IT-AE-UML-INFO-3.4
Autor	André Bauer <a(dot)bauer(at)ovm-kassel(dot)de>
Datum	18. April 2018
Links	Wikipedia: Objektorientierte Analyse und Design
Verwandte Literatur	• IT-AE-UML-INFO-3.1-3.3 • Vortragsfolien "Modellieren mit der Unified Modeling Language: Klassen- und Objektdiagramme" • Skript "Objektorientierte Programmierung mit Java und UML"
Lernjobs	IT-AE-UML-LJ-3.*
Lizenz	 Dieses Werk ist lizenziert unter einer Creative Commons Namensnennung - Weitergabe unter gleichen Bedingungen 4.0 International Lizenz.

Objektorientierte Programmierung (OOP) mit Java

Gegenben ist als Ergenis der objektorientierten Modellierung das Klassendiagramm in Abbildung 1.

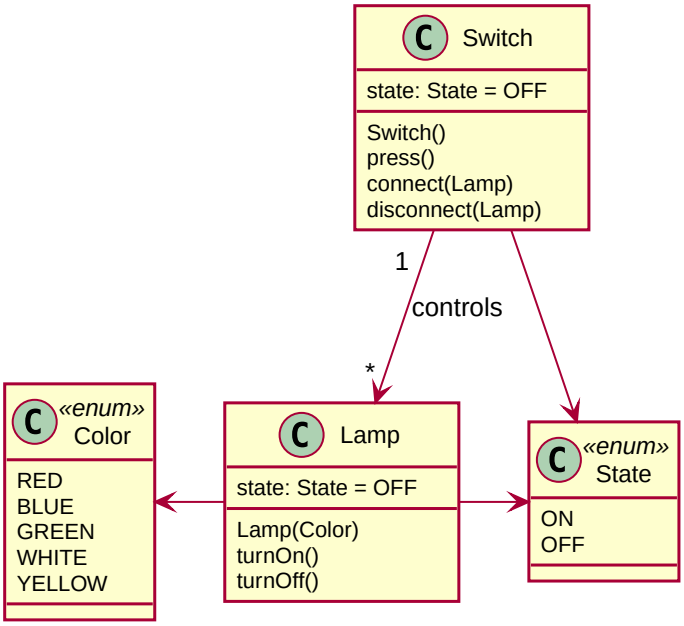

Abbildung 1. Klassendiagramm als ein Ergebnis der objektorientierten Modellierung

Dem Klassendiagramm in Abbildung 1 entspricht das folgende Java-Programm:

Quellcode 1. Java-Programm

```
// Datei State.java
public enum State {
  ON,
  OFF
}

// Datei Color.java
public enum Color {
  RED, BLUE, GREEN, WHITE, YELLOW
}

// Datei Lamp.java
public class Lamp {
  private State state;

  private Color color;

  public Lamp(Color color) {
  }

  public void turnOn() {
  }

  public void turnOff() {
  }
}

// Datei Switch.java
public class Switch {
  private State state;

  public Switch() {
  }

  public void press() {
  }

  public void connect(Lamp lamp) {
  }

  public void disconnect(Lamp lamp) {
  }
}
```

In Java muss jede Klasse mit dem Zugriffsmodifikator `public` in einer eigenen `java`-Datei mit dem Namen der Klasse gespeichert werden, so dass das Programm nun aus vier Dateien besteht.

Ist damit die Entwicklung der Software abgeschlossen? Nein, noch nicht, wir haben bislang erst

eine objektorientierte Struktur erstellt.

1. Was wird noch benötigt, damit der Schalter die Lampen steuert?

Wir benötigen ...

- Konstruktoren, die wie in den User-Stories den Anfangszustand herstellen und Objekte einer Klasse kreieren.
- eine Ausgabe, wenn die Lampe ihren Zustand wechselt, damit wir das Programm testen können.
- Quellcode, der die Beziehungen zwischen den Objekten erzeugt. Dazu betrachten wir zunächst eine Vereinfachung.
- Quellcode für die Aktionen der Objekte, wenn sie eine Nachricht erhalten, das heißt die Methoden `press()` in `Switch` sowie `turnOn()` sowie `turnOff()` in `Lamp` müssen implementiert werden.

Wir vereinfachen zunächst die Kennt-Beziehung zwischen `Switch` und `Lamp` so, dass nur eine `Lamp` mit einem `Switch` verbunden werden kann, dies nennt man eine *1-zu-1-Beziehung*.

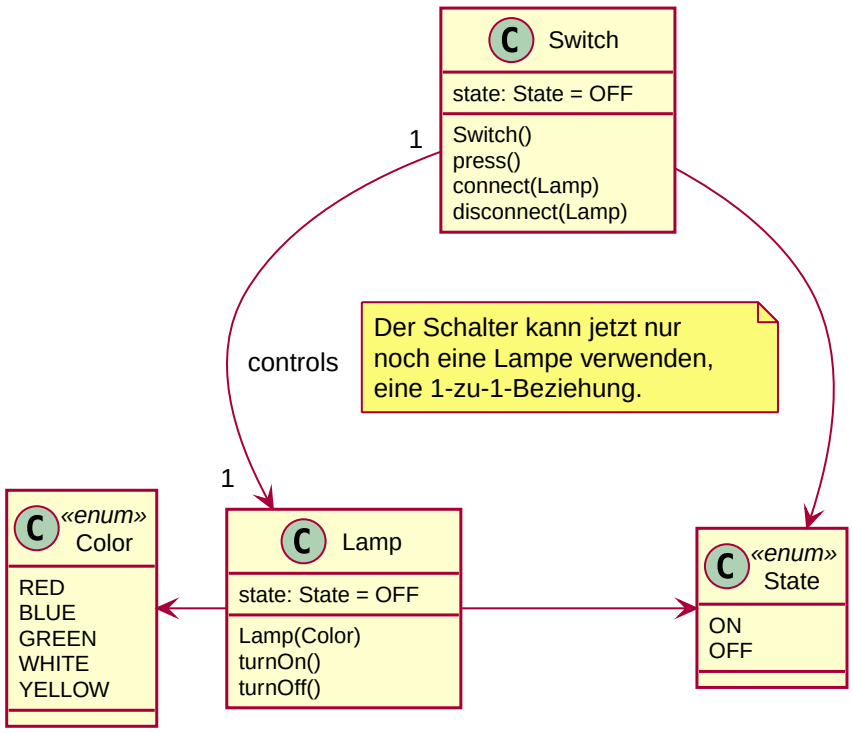

Abbildung 2. Klassendiagramm mit einer 1-zu-1-Beziehung zwischen `Switch` und `Lamp`.

Quellcode 2. Java-Programm

```
// State.java
public enum State {
  ON,
  OFF
}
```

```
// Datei Color.java
public enum Color {
  RED,
  BLUE,
  GREEN,
  WHITE,
  YELLOW
}

// Datei Lamp.java
public class Lamp {
  private State state;

  private Color color;

  public Lamp(Color color) {
    state = State.OFF;
    this.color = color;
  }

  public void turnOn() {
    state = State.ON;
    System.out.println(this + " ON"); ①
  }

  public void turnOff() {
    state = State.OFF;
    System.out.println(this + " OFF");
  }
}

// Switch.java
public class Switch {
  private State state;
  private Lamp lamp; ②

  public Switch() {
    state = State.OFF;
  }

  public void press() {
    if (lamp != null) { ③
      switch (state) {
        case OFF:
      this.state = State.ON;
          lamp.turnOn();
          break;

       case ON:
```

```
      this.state = State.OFF;
          lamp.turnOff();
        }
      }
    }

    public void connect(Lamp lampToConnect) { ④
      if (lamp == null) { ⑤
        lamp = lampToConnect;
      }
    }

    public void disconnect(Lamp lampToDisconnect) { ⑥
      if (lamp == lampToDisconnect) { ⑦
        lamp = null;
      }
    }
  }
```

① Hier wird ein Text auf der Konsole ausgegeben. Dieser setzt sich aus einer Information über das aktuellen Objekt `this` sowie dem Text `" ON"` zusammen.

② Um eine Beziehung von `Switch` zu `Lamp` herstellen zu können, wird ein Attribut der Klasse `Lamp` benötigt, dieses hat, wenn nichts angegeben wird, den initialen Wert `null`, d. h. es ist anfangs kein Objekt verknüpft.

③ Die Methoden `turnOn()` und `turnOff()` können nur aufgerufen werden, wenn `lamp` mit einem Objekt der Klasse `Lamp` verknüpft ist.

④ Die Methode `connect()` stellt die Verknüpfung einer Lampe mit einem Schalter her.

⑤ Eine neue Verknüpfung wird nur dann hergestellt, wenn der Schalter noch nicht verbunden ist.

⑥ Die Methode `disconnect()` löst eine Verknüpfung zwischen einer Lampe und einem Schalter.

⑦ Die Verknüpfung wird nur gelöst, wenn die im Parameter angegebene `lamp` mit der bereits verknüpften identisch ist.

Mit diesen Klassen können jetzt die Objekte `lamp1` und `switch1` erzeugt werden und die Lampe `lamp1` mit dem Schalter `switch1` verbunden werden.

Quellcode 3. Java-Anweisungen, um Objekte zu erzeugen und zu verbinden

```
Lamp lamp1 = new Lamp(Color.RED); ①
Switch switch1 = new Switch(); ②
switch1.connect(lamp1); ③
switch1.press(); ④
```

① Eine neues Objekt der Klasse `Lamp` mit dem Bezeichner `lamp1` wird erzeugt. Beim Aufruf des Konstruktors `Lamp(Color)` wird die Farbe der Lampe mit `Color.RED` als Parameter übergeben.

② Eine neues Objekt der Klasse `Switch` mit dem Bezeichner `switch1` wird erzeugt.

③ Die Lampe `lamp1` wird mit dem Schalter `switch1` verbunden.

④ „Betätigen“ des Schalters.

Diese Befehle können in BlueJ auch ohne `main`-Methode direkt über die Oberfläche „per Hand“ oder im integrierten CodePad ausgeführt werden. Für ein vollständiges Java-Programm wird hingegen in einer Klasse eine `main`-Methode benötigt wie in der Klasse `Controller` im Quellcode 5.

2. Wie kann man mehrere Lampen mit einem Schalter verbinden?

Wir benötigen dazu eine Objekt-Sammlung im Switch. In Java gibt es dazu vorgefertigte Komponenten, z.B. eine ArrayList. Diese hat u.a. die Methoden

`add(E e)`
: Fügt am Ende der Liste ein Objekt hinzu.

`get(int index): E`
: Gibt das Objekt an der angegeben Position `index` aus, der Index beginn in Java bei 0.

`remove(Object o)`
: Entfernt ein Objekt aus der Sammlung.

Zudem gibt es in Java eine besondere `for`-Schleife für Collections wie u. a. die ArrayList:

Quellcode 4. Java-Programm

```
for (Lamp lamp: lamps) { ①
  lamp.turnOn(); ②
}
```

① In dem Schleifenkopf wird eine Collection ausgewählt, hier `lamps` sowie ein Bezeichner `lamp` der Klasse `Lamp` definiert, der nur in dem Schleifenrumpf gültig ist.

② Im Schleifenrumpf werden die Aktionen festgelegt, die mit jedem Objekt der Sammlung durchgeführt werden, dazu verwendet man den oben definierten Bezeichner `lamp`.

Hier der vollständige Quellcode, der User-Story 1 aus IT-AE-JA-INFO-5.1 entspricht:

Quellcode 5. Vollständiges Java-Programm mit Javadoc-Kommentaren

```
// State.java
/**
 * The representation of a state with the values ON and OFF.
 *
 * @author André Bauer
 * @version 1.0
 */
public enum State {
  ON,
  OFF
}
```

```java
// Color.java
/**
 * The representation of a color.
 *
 * @author André Bauer
 * @version 1.0
 */
public enum Color {
  RED,
  BLUE,
  GREEN,
  WHITE,
  YELLOW
}

// Lamp.java
/**
 * A lamp with the states ON and OFF. It is not dimable.
 *
 * @author André Bauer
 * @version 1.0
 */
public class Lamp {
  /**
   * The current state of the lamp.
   */
  private State state = State.OFF;

  /**
   * The color of the lamp.
   */
    private Color color;

  /**
   * Creates a new lamp, with the initial state OFF.
   */
  public Lamp(Color color) {
    state = State.OFF;
    this.color = color;
  }

  /**
   * Turns the state of the lamp to ON.
   */
  public void turnOn() {
    state = State.ON;
    System.out.println(this + " ON");
  }
```

```
  /**
   * Turns the state of the lamp to OFF.
   */
  public void turnOff() {
    state = State.OFF;
    System.out.println(this + " OFF");
  }
}


// Switch.java
import java.util.*;

/**
 * A simple switch with the states ON and OFF.
 *
 * @author André Bauer
 * @version 1.0
 */
public class Switch {
  /**
   * The current state of the switch.
   */
  private State state = State.OFF;

  /**
   * The list of connected lamps.
   */
  private List<Lamp> lamps;

  /**
   * Creates a new switch witch has an initial state OFF
   * and an empty list of connected lamps.
   */
  public Switch() {
    lamps = new ArrayList<Lamp>();
  }

  /**
   * Turns the state of the switch and
   * signals every connected lamp to turn
   * its state to the state of the switch.
   */
  public void press() {
    switch (state) {
      case OFF:
    this.state = State.ON;
    for (Lamp lamp: lamps) {
          lamp.turnOn();
    }
```

```
    break;

      case ON:
    this.state = State.OFF;
    for (Lamp lamp: lamps) {
          lamp.turnOff();
    }
     }
  }

  /**
   * Connects a lamp with the switch.
   *
   * @param lamp The lamp, which should become connected with the switch.
   */
  public void connect(Lamp lamp) {
    lamps.add(lamp);
  }

  /**
   * Disconnects a lamp from the switch, if connected
   *
   * @param lamp The lamp, which should become disconnected from the switch.
   */
  public void disconnect(Lamp lamp) {
    lamps.remove(lamp);
  }
}

// Controller.java
/**
 * The main class which runs User-Story 1.
 *
 * @author André Bauer
 * @version 1.0
 */
public class Controller {
  public static void main(String args[]) {
    // User-Story 1
    Lamp lamp1 = new Lamp(Color.BLUE);
    Lamp lamp2 = new Lamp(Color.RED);
    Lamp lamp3 = new Lamp(Color.YELLOW);
    Switch switch1 = new Switch();
    switch1.connect(lamp1);
    switch1.connect(lamp2);
    switch1.connect(lamp3);
    switch1.press();
  }
}
```

Der Quellcode wird mit dem Java-Compiler `javac` übersetzt. Dabei genügt es, die Datei `Controller.java` anzugeben. Der Java-Compiler übersetzt dann automatisch alle weiteren benötigten Dateien.

```
$ javac Controller.java
```

Das Programm wird mit `java` ausgeführt:

```
$ java Controller
Lamp@6d06d69c ON ①
Lamp@7852e922 ON
Lamp@4e25154f ON
```

① `Lamp@6d06d69c` wird durch `this` in der Ausgabe mit `System.out.println(this + " ON");` erzeugt und setzt sich aus dem Namen der Klasse und der ID des Objektes zusammen.