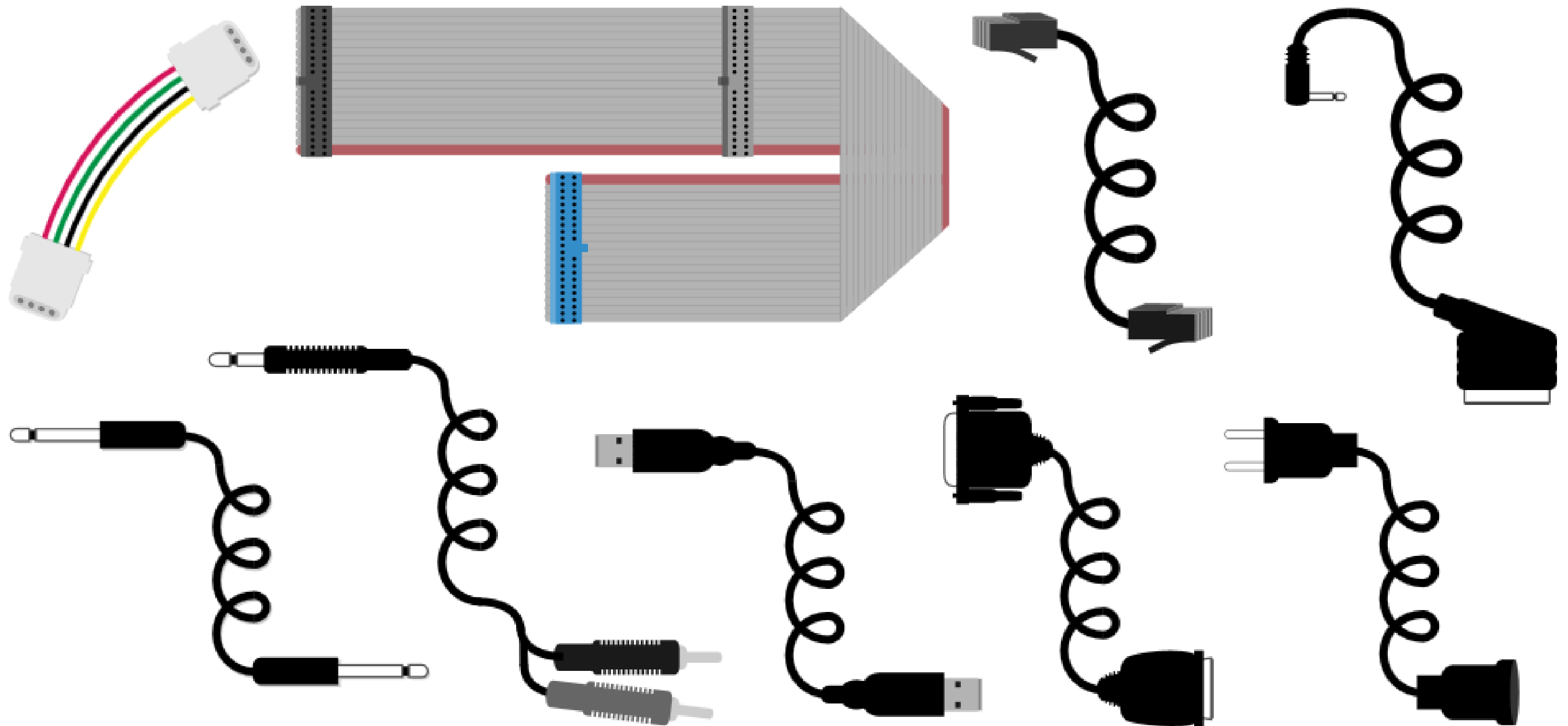
Wahlpflichtfach Design Pattern

- Dipl.-Inf., Dipl.-Ing. (FH) Michael Wilhelm
- Hochschule Harz
- FB Automatisierung und Informatik
- miwilhelm@hs-harz.de
- http://www.miwilhelm.de
- Raum 2.202
- Tel. 03943 / 659 338

Inhalt

1. Einleitung
2. Singleton
3. Observer
4. Decorator
5. Abstract Factory
6. **Adapter**
7. Facade
8. Mediator
9. Bridge
10. MVVM
11. Java Collection Framework
12. Command / Befehl



Entwurfsmuster: „Adapter“

- Dieses Pattern ist vielen Entwicklern bekannt.
- Nur kennt man es nicht unter diesem Namen. Es wird eher „intuitiv“ genutzt.

- **Minimaler Adapter:**
  - float f = 123.45f;
  - Target target = new Target();
  - Target.save2DB("Umsatz 2018",f);
    - IllegalArgument: save2DB(String, int)

```
void save2Target(String bet, float f) {
    Target target = new Target();
     target.save2DB(bez,(int) f);
}
```

Entwurfsmuster: „Adapter“

- Dieses Pattern ist vielen Entwicklern bekannt.
- Nur kennt man es nicht unter diesem Namen. Es wird eher „intuitiv“ genutzt.

- **Adapter-Pattern**
  - Es existieren zwei Klassenfamilien, die nicht kompatibel sind:
    - ArrayList vs. Array
    - HashMap vs. Array, ArrayList
    - Nicht kompatible Parameter in den Methoden
  - Abhilfe
    - Neue Klasse, die die Transformation durchführt:
      - Target: Unsere Klasse
      - Adapter Transformationsklasse
      - Adaptee Klasse, die wir benutzen möchten

Entwurfsmuster: „Adapter“

Fremde Bibliothek

Target
+Methoden

Eigene Klasse
+Methoden

Wrapperclass

Adapter
+Methoden

ev. temporär

Adaptee
+Methoden

benötigte Aktion

Adapter-Pattern

- Die Klasse Adapter wird von der Target-Klasse, der Problemklasse abgeleitet
- Einbau der „Umbaumethoden“.
- Mittels einer Komposition hat man nun Zugriff auf die Targetklasse.
- **Eine Möglichkeit!**
- Aber, man hat nicht immer Zugriff auf die fremde Klasse!!!!

1: „Adapter“ mit Ableitung

Eigene Klasse

Target
+Methoden

Test
+Methoden

Wrapperclass

Adapter
+Methoden

ev. temporär

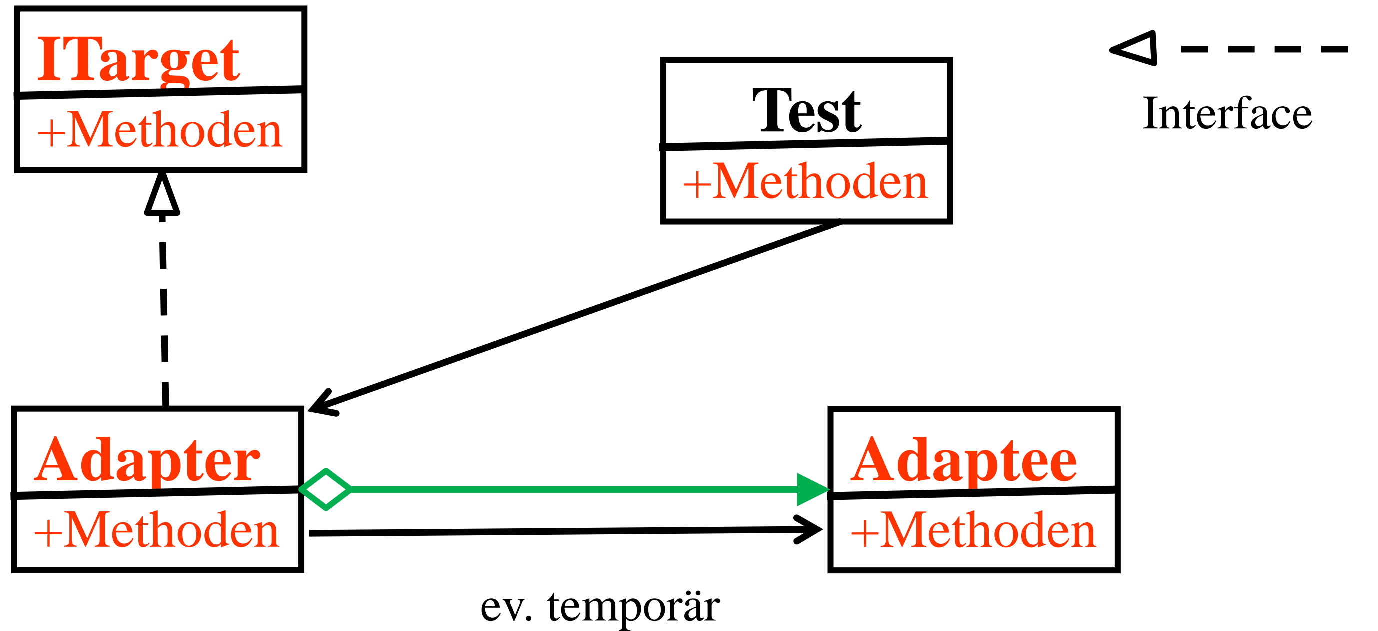
Adaptee
+Methoden

Zielklasse
Fremdklasse *.jar

Adapter-Pattern

- Die Klasse Adapter wird von der Target-Klasse abgeleitet und hat damit Zugriff auf die „Problemklasse“.
- Mittels einer Komposition hat man nun Zugriff auf die Transformation.
- Einbau aller Methoden der Adapter-Klasse um die Adaptee-Methoden aufrufen zu können.

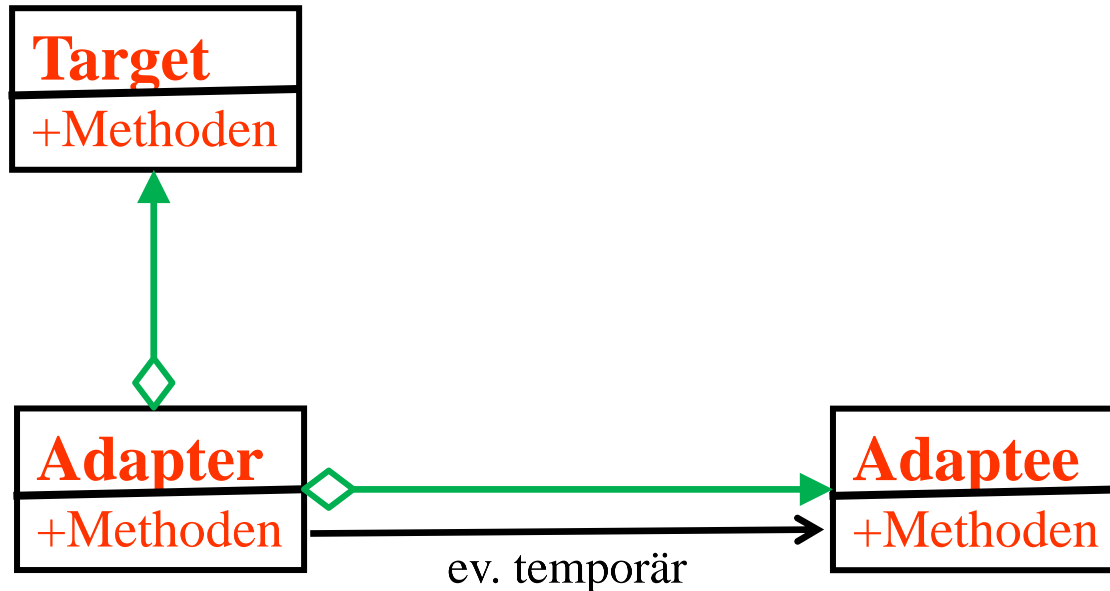
2: „Adapter“ mit Interface



Adapter-Pattern

- Die Klasse Adapter implementiert das Interface von ITarget und hat damit Zugriff auf die „Problemklasse“.
- Mittels einer Komposition hat man nun Zugriff auf die Transformation.
- Einbau aller Methoden der Adapter-Klasse um die Adaptee-Methoden aufrufen zu können.

3: „Adapter“ mit zwei Aggregationen



Adapter-Pattern

- Die Klasse Adapter hat eine Instanz der Target-Klasse und implementiert alle Methode neu (sofern gewünscht).
- Mittels einer Komposition hat man nun Zugriff auf die Targetklasse.
- Die Klasse Adapter hat eine Instanz der Adaptee-Klasse und ruft alle Methoden der Adaptee-Klasse auf.

Entwurfsmuster: „Adapter“

Wann benutzen:

- Man möchte eine „fremde“ Klasse benutzen, aber die Parameter sind unterschiedlich.
- Der Entwurf sieht möglichst eine hohe Flexibilität vor
  - Benutzung von mehreren Datenbanksystemen
- Man hat mehrere Subklassen, so dass es sinnvoller ist, die Oberklasse mit einem Adapter zu benutzen.

Weitere Patterns:

- Facade
- Bridge
- Decorator
- Proxy

Entwurfsmuster: „Adapter“

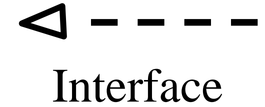
Beispiele

- Java
  - Die Hüllenklasse Integer, Double, Float, etc. in Java
  - Auto-Boxing und InBoxing bei Parametern
  - Integer result=42;

- Java-Collection:
  - java.io.InputStreamReader(InputStream)
    - returns a Reader
  - java.io.OutputStreamWriter(OutputStream)
    - returns a Writer
- WindowAdapter
  - Adapter sind vordefinierte Klassen, die alle Methoden eines Interface implementieren, damit nur noch relevante überschrieben werden müssen.
  - Beispiel: addWindowListener

1: Beispiel

Target
Eigene Klasse

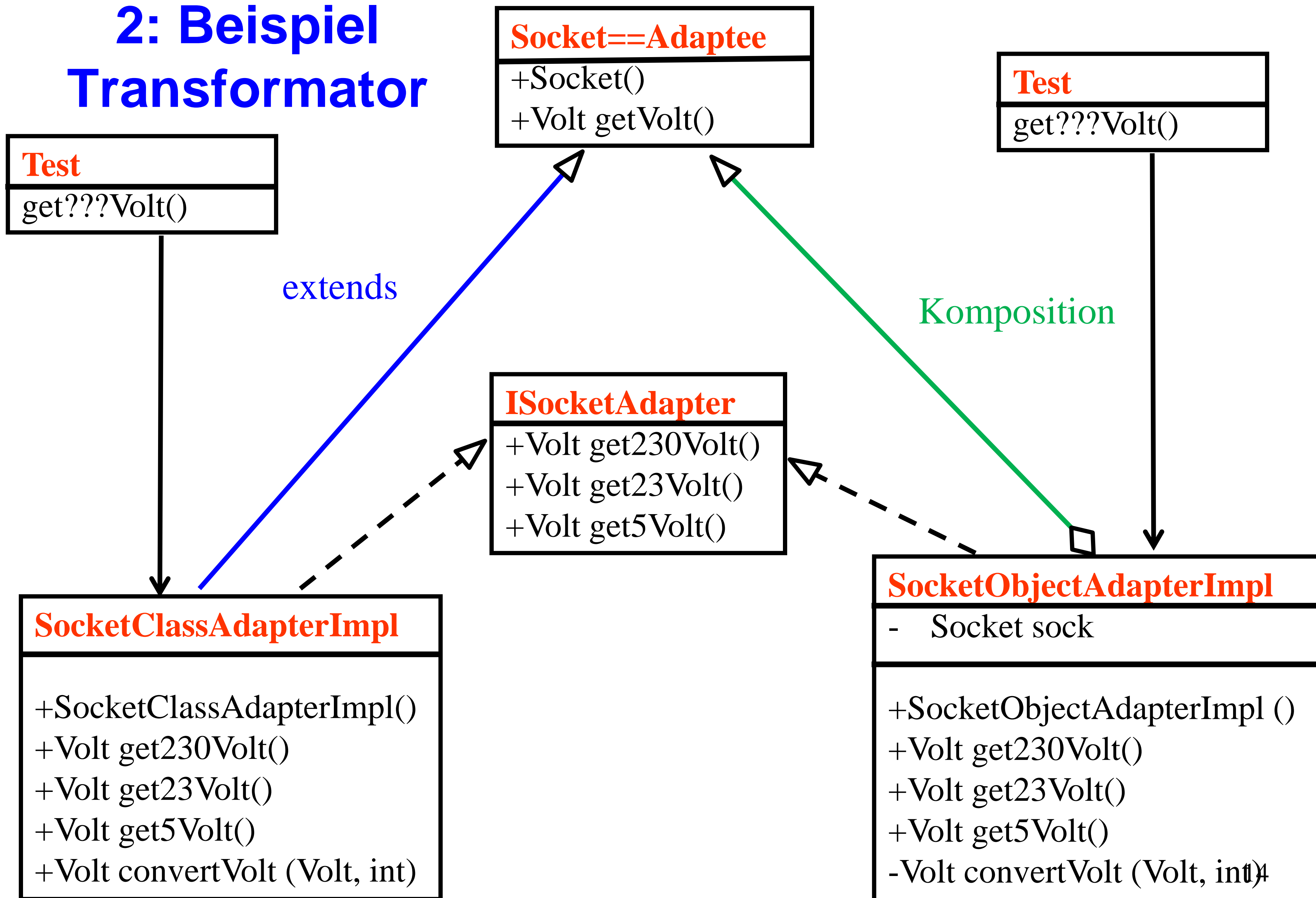


Adapter
Wrapperclass

Komposition

Adaptee
Zielklasse (jar)

2: Beispiel Transformator



2. Beispiel

- Die Klasse „Socket“ ist der „Generator“ für alle Spannungen. Diese Klasse wird immer benötigt.
  - Im linken Beispiel wird die Ableitung benutzt.
  - Im rechten Beispiel wird die Aggregation benutzt.
  - Beide Beispiele verwenden das Interface „ISocketAdapter“. Dieses dient als Kommunikation nach außen!
- Die Schnittstelle „ISocketAdapter“ stellt die Funktionalität, die Methoden für das Projekt zur Verfügung. Diese Methoden kann man jederzeit anpassen.
- Die Klassen „SocketClassAdapterImpl“ und „SocketObjectAdapterImpl“ werden nun von unserem Programm aufgerufen und benutzt.

```
public class Socket {           // Basisklasse für die Spannung
   public Volt getVolt(){
      return new Volt(230);
   }
}

public interface ISocketAdapter {

   public Volt get230Volt();

   public Volt get23Volt();

   public Volt get5Volt();
}
```

```
public class SocketClassAdapterImpl
                    extends Socket implements ISocketAdapter{
  public Volt get230Volt() {
    return getVolt();
  }

  public Volt get23Volt() {
    Volt v= getVolt();
    return convertVolt(v,10.0);
  }

  public Volt get5Volt() {
    Volt v= getVolt();
    return convertVolt(v,40);
  }
  private Volt convertVolt(Volt v, int i) {
    return new Volt(v.getVolts()/i);
  }
}
```

```
public class SocketObjectAdapterImpl implements SocketAdapter{

    //Using Composition for adapter pattern
    private Socket sock = new Socket();

    public Volt get23Volt() {
        return sock.getVolt();
    }
     public Volt get23Volt() {
        Volt v= sock.getVolt();
        return convertVolt(v,10);
    }
    public Volt get5Volt() {
        Volt v= sock.getVolt();
        return convertVolt(v,46);
    }
     private Volt convertVolt(Volt v, int i) {
        return new Volt(v.getVolts()/i);
    }
}
```

```
private void testObjectAdapter() {
    SocketAdapter sockAdapter = new SocketObjectAdapterImpl();
    Volt v5 = getVolt(sockAdapter,5);
    Volt v23 = getVolt(sockAdapter,10);
    Volt v230 = getVolt(sockAdapter,230);
    System.out.println("v5 volts using Object Adapter="+v5.getVolts());
    System.out.println("v23 volts using Object Adapter="+v23.getVolts());
    System.out.println("v230 volts using Object Adapter="+v230.getVolts());
  }
```

```
private  Volt   TestgetVolt(SocketAdapter   sockAdapter, int i) {
    switch (i){
      case     5: return sockAdapter.get5Volt();  // Aufruf mit Parameter
      case   23: return sockAdapter.get23Volt();
      case 230: return sockAdapter.get23Volt();
      default: return sockAdapter.get230Volt();
    }
  }
```

```
private void testClassAdapter() {
   SocketAdapter sockAdapter = new SocketClassAdapterImpl();
   Volt v5 = getVolt(sockAdapter,5);
   Volt v23 = getVolt(sockAdapter,10);
   Volt v230 = getVolt(sockAdapter,230);
   System.out.println("v5 volts using Class Adapter="+v3.getVolts());
   System.out.println("v23 volts using Class Adapter="+v23.getVolts());
   System.out.println("v230 volts using Class Adapter="+v230.getVolts());
}

private  Volt   TestgetVolt(SocketAdapter   sockAdapter, int i) {
   switch (i){
     case     5: return sockAdapter.get5Volt();  // Aufruf mit Parameter
     case   23: return sockAdapter.get23Volt();
     case 230: return sockAdapter.get23Volt();
     default: return sockAdapter.get230Volt();
   }
}
```

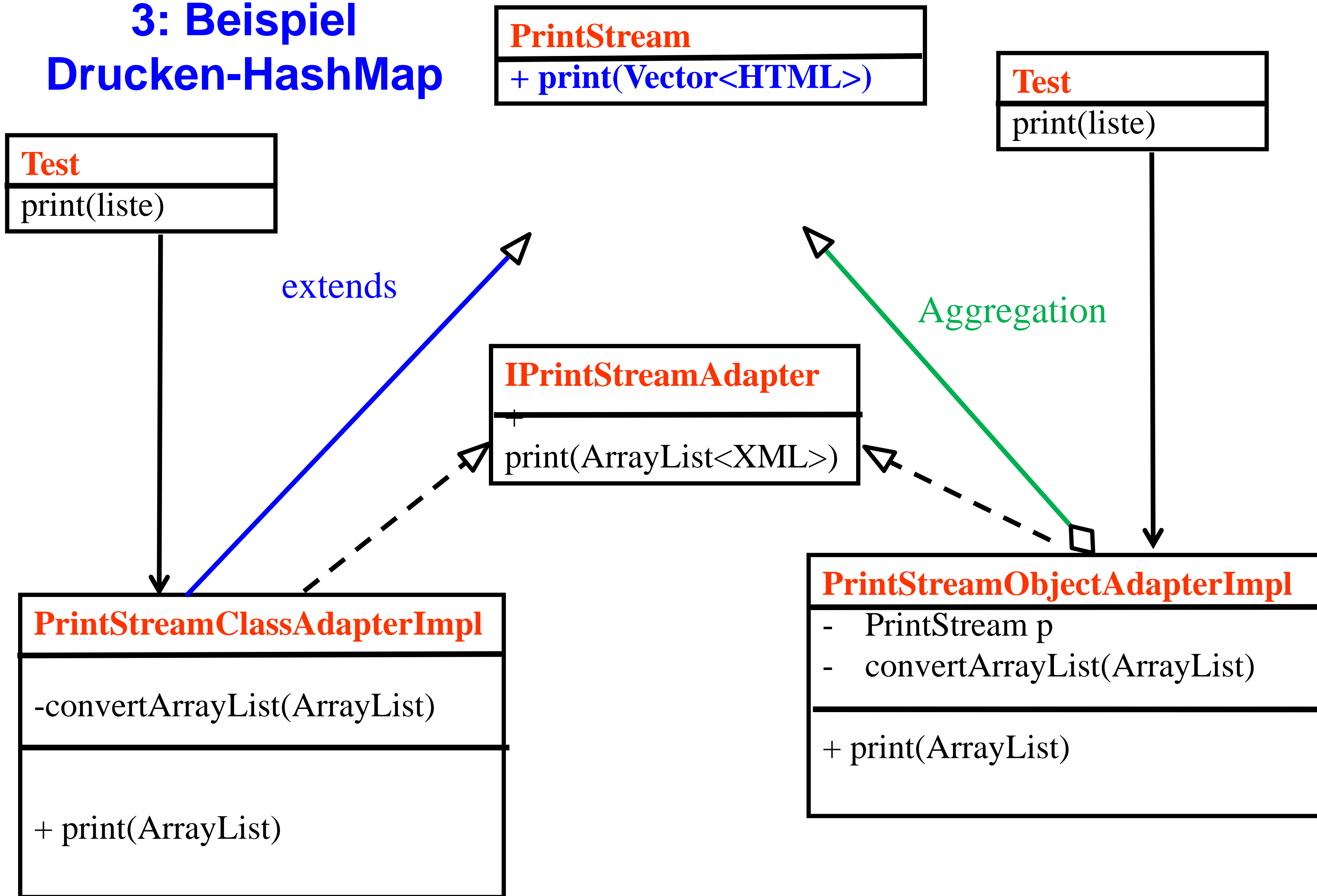
Web:

Quellen:

- http://www.codeproject.com/Articles/472801/Adapter-Design-Pattern-in-Java
- http://www.journaldev.com/1487/adapter-design-pattern-in-java-example-tutorial

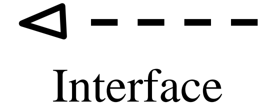
Links:

- http://www.dofactory.com/net/adapter-design-pattern
- http://code.tutsplus.com/tutorials/design-patterns-the-adapter-pattern--cms-22262
- https://msdn.microsoft.com/en-us/library/orm-9780596527730-01-04.aspx
- http://www.avajava.com/tutorials/lessons/adapter-pattern.html


3: Beispiel
Drucken-HashMap
PrintStream
+ print(Vector<HTML>)
Test
print(liste)
Test
print(liste)
extends
Aggregation
IPrintStreamAdapter
+
print(ArrayList<XML>)
PrintStreamClassAdapterImpl
-convertArrayList(ArrayList)
+ print(ArrayList)
PrintStreamObjectAdapterImpl
- PrintStream p
- convertArrayList(ArrayList)
+ print(ArrayList)

4: Beispiel Stack

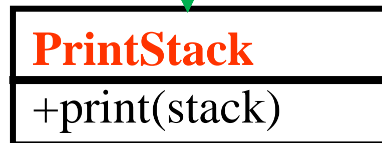
Target
Eigene Klasse



Adapter
Wrapperclass
konvertiert
Liste in Stack

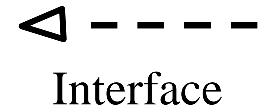
Komposition

Adaptee
Zielklasse (jar)



6. Json

Target
Eigene Klasse



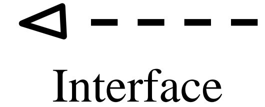
Adapter
Wrapperclass
konvertiert
Liste in Stack

Komposition

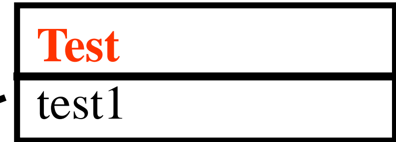
Adaptee
Zielklasse (jar)

Labor Printable

Target
Eigene Klasse



Interface



Adapter
Wrapperclass
konvertiert
Liste in Stack

Komposition

Adaptee
Zielklasse (jar)