Freie Universität Berlin

Lutz Prechelt

Softwaretechnik, SoSe21

Übung 08

TutorIn: Samuel Domiks
Tutorium 02
Materialien: Latex, Skript

Jonny Lam & Thore Brehmer

23. Juni 2023

---

1 Begriffe

(a) *Erläutern Sie jeweils knapp den wichtigsten Unterschied oder Zusammenhang zwischen ...*

1. **Schnittstelle und Signatur**
   - Eine **Schnittstelle** gibt an, welche Methoden in den unterschiedlichen Klassen vorhanden sind oder vorhanden sein müssen. [1]
   - Eine **Signatur** definiert die formale Schnittstelle einer Funktion oder Prozedur. Sie besteht aus dem Namen der Funktion sowie Parametern und Rückgabewert. [2]
2. **Klasse und Komponente**
   - Die **Klasse** dient als Bauplan und beschreibt Attribute (Eigenschaften) und Methoden (Verhaltensweisen) der Objekte.[3] Eine Klasse kann ein Modul sein.
   - **Komponente** in 3.
3. **Komponente und Modul**
   - Ein **Modul** (neuerdings auch **Komponente** genannt) ist ein hierarchisches Teil vom System. Ein Modul ist also eine Einheit zusammengehörender Programmelemente (z.B. Daten, Datentypen, Klassen, Funktionen etc). Auf den oberen Ebenen nennt man es auch Subsystem. (Module sollten möglichst geringe Kopplung mit anderen Modulen haben.)
4. **Kohäsion und Kopplung**
   - **Kohäsion und Kopplung** sind ein Maß um die Komplexität ihrer Software zu analysieren. Es beschreibt die Komplexität der Beziehungen zwischen den Klassen und die innere Zusammengehörigkeit innerhalb der Klassen.[6]
   - **Kohäsion** beschreibt wie gut eine Programmeinheit eine logische Aufgabe oder Einheit abbildet. In einem System mit starker Kohäsion ist jede Programmeinheit (eine Methode, eine Klasse oder ein Modul) verantwortlich für genau eine wohldefinierte Aufgabe oder Einheit. [4]

- Unter **Kopplung** versteht man die Verknüpfung von verschiedenen Systemen, Anwendungen oder Softwaremodulen sowie ein Maß, das die Stärke dieser Verknüpfung bzw. der daraus resultierenden Abhängigkeit beschreibt.[5]

(b) *Erklären Sie die offensichtlichsten Unterschiede zwischen Komponenten, Entwurfsmustern und Architekturstilen.*

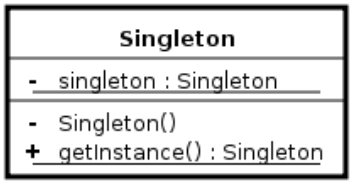
- Ein **Architekturstil** beschreibt welche Teile das System hat, wie diese zusammenspielen und wie dadurch die Funktionalen und nicht Funktionalen Anforderungen erfüllt werden.
- **Komponenten** sind Teile des Systems, welche durch zerlegen des Systems entstehen.
- **Entwurfsmuster** sind bewährte Lösungsschablonen für wiederkehrende Entwurfsprobleme/-Teilprobleme eines Systems in der Softwarearchitektur.[12]

(c) *Recherchieren Sie das Entwurfsmuster Einzelstück (engl.singleton). Denken Sie wie immer daran Ihre Quellen anzugeben.*

1. *Erläutern Sie das Entwurfsmuster, d.h. welches Problem löst es wie?*

   - Das **Singleton** findet Verwendung, wenn nur ein Objekt zu einer Klasse existieren darf und ein einfacher Zugriff auf dieses Objekt benötigt wird (oder das einzige Objekt durch Unterklassenbildung spezialisiert werden soll.)[7]

2. *Illustrieren Sie das Entwurfsmuster indem Sie drei hypothetische Beispiele für seine Verwendung formulieren.*



[7]

   - **Verwendungs Beispiele:**
     1. Für die Darstellung einer Zentralbank.
     2. Für die Darstellung eines Königs.
     3. Für die Darstellung eines Gottes (für monotheistische Religionen).
        (Da es für jedes dieser Beispiele höchstens ein Objekt geben kann)

3. *Zu welcher Klasse der in der Vorlesung vorgestellten Entwurfsmustertaxonomie (pattern taxonomy) gehört dieses Muster und warum?*

   - Da beim **Singleton** von einer Klasse nur ein einziges Objekt erzeugt werden darf, passt es gut zum **Erzeugungsmuster**, da das Ziel des Erzeugungsmuster ist das System unabhängig davon zu machen wie seine Objekte erzeugt, komponiert und repräsentiert werden.

(d) *Charakterisieren und vergleichen Sie die folgenden Entwurfsmuster: Stellvertreter(Proxy), Adapter (Adapter), Fassade (Facade), Brücke (Bridge).*

- **Stellvertreter(Proxy):** Umschließt ein anderes Objekt und kontrolliert den Zugriff darauf.[14]
- **Adapter (Adapter):** Umschließt ein Objekt(Schnittstelle) und stelle eine andere Schnittstelle dafür zur Verfügung. Entweder weil der Client sonst nicht darauf zugreifen kann, oder um die Schnittstelle zu vereinfachen. Dadurch muss weder beim Client und noch beim Adaptee was verändert werden, nur beim Adapter.[13]
- **Fassade (Facade):** Vereinfacht eine Schnittstelle.[15]

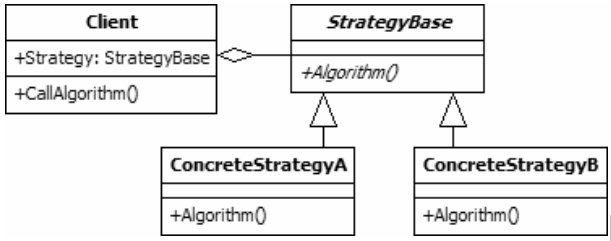
- **Brücke (Bridge)** Erlaubt separate Entwicklung von Abstraktions- und Implementierungshierarchien.[16]

2 Entwurfsmuster für eigene Software-Idee

(a) *Überlegen Sie sich, welches der in der Vorlesung vorgestellten Entwurfsmuster Sie in Ihrer zu entwickelnden Software sinnvoll einsetzen können (außer dem Singleton und dem Adapter). Stellen Sie die Charakteristika des gewählten Entwurfsmusters heraus (d.h. welches Problem löst es wie) und begründen Sie, warum und wofür das Muster in Ihrem Kontext geeignet ist.*

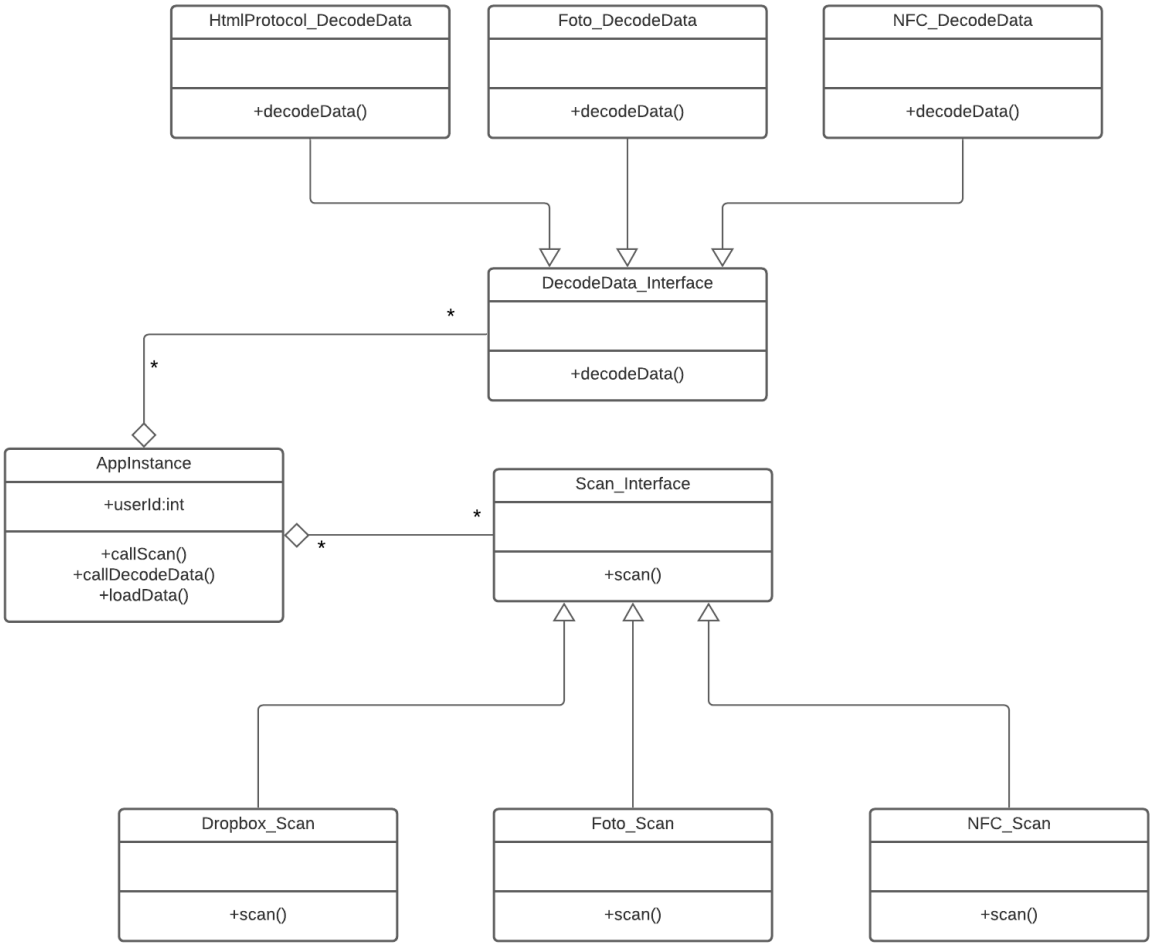
- Das **Strategy** Entwurfsmuster wird verwendet, um eine austauschbare Familie von Algorithmen zu erstellen, aus der der erforderliche Prozess zur Laufzeit ausgewählt wird. Dadurch kann sich das Verhalten eines Programms je nach Konfigurationsdetails oder Benutzerpräferenzen dynamisch ändern. Es erhöht auch die Flexibilität, indem es ermöglicht, dass neue Algorithmen in Zukunft einfach integriert werden können.[11]
- **Es löst also folgende Probleme**: "Wenn man verschiedene Wege hat, etwas zu machen, dann soll man diese frei hinzufügen, benutzen oder austauschen können, je nachdem, was gerade gebraucht wird."[10]
- **Für unseren Kontext ist es geeignet**, da wir mit unseren Digitalen Bon App Bons mithilfe von verschiedenen Verfahren einscannen wollen. Die verschiedenen Scans benötigen unterschiedliche Scan- sowie Decode Methoden (Scan zum einlesen der Daten und Decode um die Daten auch verwenden zu können). Das Ergebnis von verschiedenen Scan Methoden könnten aber von einer einzigen Decode Methode benutzt werden. (Bsp. Das Ergebnis von Scan Dropbox und Scan GoogleDrive kann von DecodeHtml benutzt werden)
- Um gute Skalierbarkeit (vielleicht werden zukünftig neue Scan Methoden verwendet) und gute Code leserlichkeit zu ermöglichen (anstatt für jeden Scan+Decode eine einzelne neue Klasse) haben wir uns für das Strategy Entwurfsmuster entschieden.

(b) *Stellen Sie das Entwurfsmuster in seiner allgemeinen Form, d.h. unter Verwendung der Rollennamen, in UML-Notation dar. Recherchieren Sie nach Bedarf und geben Sie in jedem Fall Ihre Quellen an.*



[11]

(c) *Übertragen Sie nun das gewählte Entwurfsmuster in Ihren Kontext und stellen das Ergebnis ebenfalls als UML-Diagramm dar.*



(d) *Nicht alle Aspekte eines Entwurfsmusters lassen sich bequem in einem UML-Diagrammausdrücken. Implementieren Sie (in Java oder Pseudocode) in Ansätzen die Verwendung des Entwurfsmusters in Ihrem Kontext. Erstellen Sie die für das Muster notwendigen Schnittstellen und Klassen, mit den jeweils relevanten Code stellen, d.h. zumindest angedeutete Daten- und Kontrollstrukturen (leere Methodenrümpfe reichen in aller Regel nicht aus). Schreiben Sie ausdrücklich keinen Code, der über das bloße Entwurfs-muster hinaus geht!*

```java
package task_2n;

public class AppInstance {
 int userID;
 Scan_Interface scanI;
 DecodeData_Interface decodeI;

 public AppInstance(int userID, Scan_Interface scanI, DecodeData_Interface decodeI)
     {
  this.userID = userID;
  this.scanI = scanI;
  this.decodeI = decodeI;
 }

 public void loadData(String selected) {
  byte[] data = scanI.scan();
  data = decodeI.decodeData(data);
  //maybe some further calculation with data//
  }
}
```

```java
package task_2n;

public interface Scan_Interface {
 public byte[] scan();
}
```

```java
package task_2n;

public class NFC_Scan implements Scan_Interface{

 public byte[] scan() {
  // byte[] data = load data with NFC protocol
  // return data
 }

}
```

```java
package task_2n;

public class Foto_Scan implements Scan_Interface {

 public byte[] scan() {
  // byte[] data = load data from a foto scan (e.g. Barcode or QR-Code)
  // return data
 }

}
```

```java
package task_2n;

public class Dropbox_Scan implements Scan_Interface {

 public byte[] scan() {
  // byte[] data = load data from a dropbox
  // return data
 }

}
```

```java
package task_2n;

public interface DecodeData_Interface {
 public byte[] decodeData(byte[] data);
}
```

```java
package task_2n;

public class NFC_DecodeData implements DecodeData_Interface {

 public byte[] decodeData(byte[] data) {
  // data = translate data in a use able form with the nfc protocol
  // return data
 }

}
```

```java
package task_2n;

public class Foto_DecodeData implements DecodeData_Interface {

 public byte[] decodeData(byte[] data) {
  // data = translate data in a use able form with a foto (e.g. Barcode QR-Code)
     protocol
  // return data
 }

}
```

```java
package task_2n;

public class HtmlProtocol_DecodeData implements DecodeData_Interface {

 public byte[] decodeData(byte[] data) {
  // data = translate data in a use able form with the html protocol
  // return data
 }

}
```

3 : Adapter-Entwurfsmuster anwenden

(a) ***Aufgabe:*** *Implementieren Sie eine neue Klasse (etwa „Storage“) mit einer einzigen Methode (etwa „store“), die Dateinamen und Inhalt erhält und Ihre alte Filesystem-Klasse wiederverwendet – ohne diese zu verändern!*

```
package u8;

public class Storage {
 private Filesystem filesystem;

 public Storage(Filesystem filesystem) {
  this.filesystem = filesystem;
 }

 public void store(String filename, String content_str) {
  filesystem.remove(filename);
  filesystem.touch(filename);
  int sector = filesystem.position(filename);
  byte[] content = content_str.getBytes();
  filesystem.write(sector, content);
 }
}
```

```
package u8;

public class NotepadMinusMinus {
 /* ... */

 public void save() {
  String content = DocBuffer.getContent();
  String name = DocManager.getCurrentFilename();

  Filesystem filesystem = new Filesystem();
  Storage storage = new Storage(filesystem);
  storage.store(name, content);
 }
}
```

(b) *Das klassische Adapter-Muster kennt die folgenden vier Rollen: Client, Target, Adapter und Adaptee. Geben Sie für jede dieser Rollen an, durch welche der Klassen Ihrer Implementierung diese ausgefüllt wird?*

- Client →NotepadMinusMinus
- Target →None
- Adapter →Storage
- Adaptee →Filesystem

(c) *Nutzen Sie das Dropbox-SDK, um Ihren Editor um eine Speicher-Option zu erweitern. Wie schon bei der Filesystem-Wiederverwendung sollen (bzw. können) Sie den Dropbox-Quellcode nicht verändern. Schreiben Sie also einen weiteren Adapter hierfür. Um in der NotepadMinusMinus-Klasse nicht direkt von einem der beiden Adapter (einen für das lokale Dateisystem, einen für Dropbox) abzuhängen, führen Sie bitte ein Interface ein (etwa „IStorage“), das die beiden Adapter jeweils implementieren. Um in der NotepadMinusMinus-Klasse davon Gebrauch machen zu können, implementieren Sie bitte eine Setter-Methode und verwenden den übergebenen Wert in der bereits existierenden save()-Methode:*

```java
package u8;

public interface IStorage {
 public void store(String filename, String content_str);
}
```

```java
package u8;

public class Storage implements IStorage{
 private Filesystem filesystem;

 public Storage(Filesystem filesystem) {
  this.filesystem = filesystem;
 }

 public void store(String filename, String content_str) {
  filesystem.remove(filename);
  filesystem.touch(filename);
  int sector = filesystem.position(filename);
  byte[] content = content_str.getBytes();
  filesystem.write(sector, content);
 }
}
```

```java
package u8;

public class StorageDropbox implements IStorage{
 private DbxClient dropbox;

 public StorageDropbox(DbxClient dropbox) {
  this.dropbox = dropbox;
 }

 public void store(String filename, String content_str) {
  byte[] content = content_str.getBytes();
  if(dropbox.fileExists(filename)) {
   dropbox.uploadFile(filename, DbxClient.WriteMode.REPLACE, content);
  }
  else {
   dropbox.uploadFile(filename, DbxClient.WriteMode.ADD, content);
  }
 }
}
```

```java
package u8;

public class NotepadMinusMinus {
 /* ... */
 IStorage storage;

 public void save() {
  String content = DocBuffer.getContent();
  String name = DocManager.getCurrentFilename();


  setStorage(new Storage(new Filesystem()));
  //or
  setStorage(new StorageDropbox(new DbxClient()));
  storage.store(name, content);
 }

 public void setStorage(IStorage s) {
  this.storage = s;
 }
}
```

(d) *Analog zu Aufgabe b): Wie sieht die Rollenverteilung (also Client, Target, Adapter, Adaptee) auf die Klassen und Schnittstellen der zweiten Implementierung aus? Bedenken Sie, dass Sie nun zwei Exemplare des Adapter-Musters vorliegen haben.*

- Client →NotepadMinusMinus
- Target →IStorage
- Adapter →Storage, StorageDropbox
- Adaptee →Filesystem, DbxClient

(e) *In der zweiten Implementierung „weiß" die zentrale Editor-Klasse zur Kompilierzeit nicht, welche Implementierung des Storage-Interfaces sie konkret benutzen wird. Diese wird erst zur Laufzeit über die setStorage()-Methode „hineingereicht".Recherchieren Sie die Idee der „Dependency Injection" (Quellen angeben). Welche Vorteile halt dieses Entwurfsprinzip? Welche Nachteile?*

- Als **Dependency Injection** wird in der objektorientierten Programmierung ein Entwurfsmuster bezeichnet, welches die Abhängigkeiten eines Objekts zur Laufzeit reglementiert: Benötigt ein Objekt beispielsweise bei seiner Initialisierung ein anderes Objekt, ist diese Abhängigkeit an einem zentralen Ort hinterlegt – es wird also nicht vom initialisierten Objekt selbst erzeugt.[8]

  Vorteile [9]:

- **Flexibilität:** Nur das Verhalten des Clients ist festgelegt, der Client kann auf alles reagieren was die vom Client erwartete interne Schnittstelle unterstützt.
- **Systeme können ohne Neukompilierung neu konfiguriert werden**, da die Konfigurationsdaten ein Konfigurationsdateien ausgelagert werden können.
- **Wiederverwendbarkeit,Testbarkeit und Wartbarkeit**, da man eine ganze Implementierung entfernen kann um Auswirkungen von Designänderung und fehlern zu isolieren.
- **Gleichzeitige oder unabhängige Entwicklung**, da Entwickler nur die Schnittstelle kennen müssen um unabhängig voneinander Klassen zu entwickeln.
  Nachteile[9]:
- **Lesen von Code wird erschwert**, da das Verhalten von der Konstruktion getrennt wird. Entwickler müssen auf weitere Dateien verweisen um die Leistung eine Systems zu verfolgen.
- **Mehr Entwicklungsaufwand im Voraus erforderlich.** Man weiß im Vorhinein nicht wann und wo es benötigt wird, sondern man muss anfragen dass es injiziert wird und dann sicherstellen, dass es wirklich injiziert wurde.
- **Schwierig in die Verknüpfung zwischen Klassen zu gelangen.**

4 Quellen

[1] "Schnittstelle (Objektorientierung)", https://de.wikipedia.org/wiki/Schnittstelle_(Objektorientierung) . [aufgerufen am 01.06.2021]

[2] "Signatur (Programmierung)", https://de.wikipedia.org/wiki/Signatur_(Programmierung) [aufgerufen am 01.06.2021]

[3] "Klasse (Objektorientierung)", https://de.wikipedia.org/wiki/Klasse_(Objektorientierung) [aufgerufen am 01.06.2021]

[4] "Kohäsion (Informatik)", https://de.wikipedia.org/wiki/Kohäsion_(Informatik) [aufgerufen am 01.06.2021]

[5] "Kopplung (Softwareentwicklung)", https://de.wikipedia.org/wiki/Kopplung_(Softwareentwicklung) [aufgerufen am 01.06.2021]

[6] “Kohäsion und Kopplung”, https://sites.google.com/site/koesterprogramming/home/softwareentwicklung/kohaesion-und-kopplung [aufgerufen am 01.06.2021]

[7] “Singleton (Entwurfsmuster)”, https://de.wikipedia.org/wiki/Singleton_(Entwurfsmuster) [aufgerufen am 01.06.2021]

[8] “Dependency Injection” https://de.wikipedia.org/wiki/Dependency_Injection [aufgerufen am 04.06.2021]

[9] “Dependency Injection Vorteile_und_Nachteile” https://de.wikipedia.org/wiki/Dependency_Injection#Vorteile_und_Nachteile [aufgerufen am 04.06.2021]

[10] “Vorlesung 12”, (page 26/42)

[11] “Strategy Design Pattern”, http://www.blackwasp.co.uk/Strategy.aspx [aufgerufen am 05.06.2021]

[12] “Entwurfsmuster”, https://de.wikipedia.org/wiki/Entwurfsmuster [aufgerufen am 05.06.2021]

[13] “Adapter Design Pattern”, http://www.blackwasp.co.uk/Adapter.aspx [aufgerufen am 05.06.2021]

[14] “Proxy Design Pattern”, http://www.blackwasp.co.uk/Proxy.aspx [aufgerufen am 05.06.2021]

[15] “Facade Design Pattern”, http://www.blackwasp.co.uk/Facade.aspx [aufgerufen am 05.06.2021]

[16] “Bridge Design Pattern”, http://www.blackwasp.co.uk/Bridge.aspx [aufgerufen am 05.06.2021]