Freie Universität Berlin, Institut für Informatik, Arbeitsgruppe Software Engineering
Barry Linnert, Victor Brekenfeld, Tamara Fischer, Samuel Domiks, Abraham Söyler

Übungsblatt 8 **Entwurfsmuster** zum 2021-06-07

Aufgabe 8-1: Begriffe

Lernziele: Zentrale Konzepte aus dem Bereich des Softwareentwurfs erklären können, geläufige Entwurfsmuster kennen.

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

1. Schnittstelle und Signatur
2. Klasse und Komponente
3. Komponente und Modul
4. Kohäsion und Kopplung

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

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?
2. Illustrieren Sie das Entwurfsmuster indem Sie drei hypothetische Beispiele für seine Verwendung formulieren.
3. Zu welcher Klasse der in der Vorlesung vorgestellten Entwurfsmustertaxonomie (*pattern taxonomy*) gehört dieses Muster und warum?

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

Aufgabe 8-2: Entwurfsmuster für eigene Software-Idee

Lernziele: Entwurfsmuster geeignet auswählen und anwenden können; Entwurfsmuster von dessen konkreter Anwendung unterscheiden können.

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.

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.

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-Diagramm ausdrücken. 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 Codestellen, d.h. zumindest angedeutete Daten- und Kontrollstrukturen (leere Methodenrümpfe reichen in aller Regel *nicht* aus). Schreiben Sie ausdrücklich keinen Code, der über das bloße Entwurfsmuster hinaus geht!

Denken Sie wie immer daran, Ihre Arbeitsergebnisse der Teilaufgaben **a)** bis **d)** zusätzlich zur elektronischen Abgabe über das KVV Ihrer Wiki-Seite hinzuzufügen.

Aufgabe 8-3★: Adapter-Entwurfsmuster anwenden

Lernziele: Eines der wichtigsten Entwurfsmuster in einem realistischen Kontext konkret handwerklich umsetzen können.

Vorbemerkung: Lassen Sie sich nicht vom vielen Text dieser Aufgabe abschrecken. Inhaltlich ist sie weder schwer noch aufwendig.

Hintergrund: Stellen Sie sich vor, Sie sind dabei einen einfachen Texteditor mit dem Namen „Notepad−−“ in Java zu entwickeln. Der Funktionsumfang ist sehr reduziert, und Sie haben alle Hände voll zu tun, diesen zu erweitern. Daher wollen Sie, so gut es geht, existierende Lösungen wiederverwenden.

a) Für die Dateiverwaltung Ihres Editors wollen Sie eine Klasse `Filesystem` wiederverwenden, die Sie früher einmal entwickelt haben. Diese funktioniert tadellos, und deswegen wollen Sie die auch nicht mehr ändern. Sie haben sich daher für das Adapter-Entwurfsmuster entschieden.

Dies ist ein Auszug aus der Schnittstellendokumentation der `Filesystem`-Klasse:

```
public class Filesystem {
    // Returns the ID of the logical sector where the given file resides.
    // Returns -1 if no such file exists.
    public int position(String filename) { /* ... */ }

    // Write the content on the disk into the sector with the given ID
    public void write(int sector, byte[] content) { /* ... */ }

    // Prepares the creation of a new file. Provide the content
    // through the write method! Fails if the name is already taken.
    public void touch(String filename) { /* ... */ }

    // Erases a file, and compacts the content of the disk, such that
    // there are no gaps in between.
    public void remove(String filename) { /* ... */ }
}
```

Der Gebrauch der `Filesystem`-Klasse ist etwas ungelenkt, und in Ihrer Hauptklasse `NotepadMinusMinus` wollen Sie lediglich einen einzigen Aufruf zum Speichern des aktuellen Editor-Inhalts vornehmen. Folgendes Gerüst haben Sie bereits implementiert:

```
public class NotepadMinusMinus {
    /* ... */

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

        // TODO Save to disk (replace existing file, if any)
    }
}
```

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!

Ergänzen Sie außerdem die `save()`-Implementierung aus obigen Fragment, um die `Storage`-Klasse zu benutzen. Führen Sie keine weiten Klassen oder Schnittstellen ein.

b) Das klassische Adapter-Muster kennt die folgenden vier Rollen: Client, Target, Adapter und Adaptee.[1] Geben Sie für jede dieser Rollen an, durch welche der Klassen Ihrer Implementierung diese ausgefüllt wird?

[1] siehe z.B. http://www.blackwasp.co.uk/Adapter.aspx

c) Nun wollen Sie Ihrem Editor auch eine Dropbox-Integration spendieren. Für solche Zwecke bietet Dropbox selbst ein SDK (eine Bibliothek) an. Gehen Sie für diese Aufgabe von folgender vereinfachter API (Programmierschnittstelle) aus:

```
public class DbxClient {
    public enum WriteMode {
        ADD, REPLACE
    }

    // Checks whether a file already exists on the server
    public boolean fileExists(String name) { /* ... */ }

    // Uploads a new version of a file; either as a new file or as a
    // replacement for an existing one. New files must be transmitted
    // using the ADD mode, replacements must be done in REPLACE mode.
    // Incorrect modes indicate inconsistencies and the upload will
    // fail.
    public void uploadFile(String name, WriteMode mode, byte[] data) {
        /* ... */ }
}
```

Aufgabe: 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 (siehe unten) und verwenden den übergebenen Wert in der bereits existierenden `save()`-Methode:

```
public class NotepadMinusMinus {
    /* ... */

    public void setStorage(IStorage 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.

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?