— Übungsblatt 2 (T): Entwurfsmuster —

Aufgabe 1 (Entwurfsprinzipien) (T)

1. Was ist das Offen/Geschlossen-Prinzip? Welche Nachteile könnten entstehen, wenn das Prinzip verletzt wird?
2. Was ist `Inversion of Control`? Überlegen Sie sich zwei Vorteile, die dieses Entwurfsprinzip bieten könnte.
3. Warum ist es besser gegen Schnittstellen anstelle von Implementierungen zu programmieren?
4. Wie wirken sich die in der Vorlesung besprochenen Entwurfsprinzipien auf die Testbarkeit einer Anwendung aus?

Aufgabe 2 (Entwurfsmuster) (T)

1. Erklären Sie den Unterschied zwischen einfachen Heuristiken und Entwurfsmustern. Welche Vorteile ergeben sich aus beiden Ansätzen?
2. Was sind Entwurfsmuster? Wie entstehen sie und wie werden sie dokumentiert?
3. Was ist eine `Fassade`?
4. Beschreiben Sie das Entwurfsmuster `Singleton`.
5. Wie funktioniert das `Proxy`-Muster und wofür kann es verwendet werden?
6. Skizzieren Sie ein Klassendiagramm des Musters `Observer`.

Aufgabe 3 (Noch mehr Entwurfsmuster) (T)

a) Das Klassendiagramm eines Dekorierers sieht genauso aus, wie das Klassendiagramm eines Proxys, jedoch soll ein Dekorierer ein vorhandenes Objekt einhüllen, um dessen Methoden mit zusätzlicher Funktionalität auszustatten. Nennen Sie zwei Stellen in der Ihnen bekannten Java API, wo Sie dieses Muster bereits kennengelernt haben.

b) Zeigen Sie, wie das Proxymuster die Entwicklung von verteilten Anwendungen vereinfachen kann. Erklären Sie auch, warum das Muster den Entwurf tatsächlich vereinfacht und nicht komplizierter macht.

c) Skizzieren Sie ein allgemeines Klassendiagramm des MVC-Musters und beschreiben Sie dann, welche GoF-Muster darin Verwendung finden. Gehen Sie dabei auch auf mögliche Variationen und die unterschiedlichen Einsatzgebiete des Musters ein. Welche Variationen können Sie sich vorstellen und welche Vor- und Nachteile würden diese bringen?

Aufgabe 4 (Code-Magneten) (P)

Sie haben gerade Ihre neuste, mustergültige Anwendung fertiggestellt, als ein Bösewicht die ganzen Quellcodes zerschnippelt und als Scherz an den Kühlschrank geklebt hat. Zum Glück konnten Sie die meisten Klassen bereits rekonstruieren, doch eine Klasse, die das Beobachtermuster implementiert, fehlt Ihnen noch.

Ordnen Sie die Schnippsel in der richtigen Reihenfolge an. Da Sie beim Staubsaugen aus Versehen einige der geschweiften Klammern eingesaugt haben, müssen Sie diese an den passenden Stellen ergänzen.

Schnippsel 1

```
public MusicDatabase() {
```

Schnippsel 2

```
subscribers = new ArrayList<Subscriber>();
```

Schnippsel 3

```
public void readData() {
```

Schnippsel 4

```
// Hier wird die Datenbank eingelesen
```

Schnippsel 5

```
public class MusicDatabase
```

Schnippsel 6

```
// Hier passiert noch nichts, weil es noch gar keine
// Beobachter gibt.
```

Schnippsel 7

```
import java.util.ArrayList;
import java.util.List;
```

Schnippsel 8

```
private void notifySubscribers() {
    for(Subscriber s : this.subscribers) {
        s.somethingHappened(this);
    }
}
```

Schnippsel 9

```
private List<Subscriber>
```

Schnippsel 10

```
    this.subscribers.remove(s);
}
```

Schnippsel 11

```
public void addSubscriber(Subscriber s) {
```

Schnippsel 12

```
implements Publisher {
```

Schnippsel 13

```
// Hier wird die Datenbank aktualisiert
```

Schnippsel 14

```
this.notifySubscribers();
```

Schnippsel 15

```
this.notifySubscribers();
```

Schnippsel 16

```
if (!this.subscribers.contains(s)) {
    this.subscribers.add(s);
}
```

Schnippsel 17

```
public void addTitle(String title, double duration) {
```

Schnippsel 18

```
if (this.subscribers.contains(s)) {
```

Schnippsel 19

```
public void removeSubscriber(Subscriber s) {
```