UNIVERSITÄT
LEIPZIG

Softwaretechnik 2024/25 – Übung 07

Prof. Dr. Norbert Siegmund
B. Sc. Annemarie Wittig

Aufgabe 1: Grundlagen

a) Was ist Qualität

- Interne Qualität:
  - Erweiterbarkeit, Wartbarkeit, Verständlichkeit, Lesbarkeit
  - Robust gegenüber Änderungen
  - Kopplung und Kohäsion
  - Wiederverwendbarkeit

  → Zusammengefasst typischerweise beschrieben als Modularität

- Externe Qualität:
  - Korrektheit: Erfüllung der Anforderungen
  - Einfachheit in der Benutzung
  - Ressourcenverbrauch
  - Legale und politische Beschränkungen

Aufgabe 1: Grundlagen

b) Nennen und erläutern Sie die fünf Kriterien für gutes Design.

- Modular Decomposability
- Modular Composability
- Modular Understandability
- Modular Continuity
- Modular Protection

Aufgabe 1: Grundlagen

c) Nennen und erläutern Sie die fünf Regeln für gutes Design.

- Direct Mapping
- Few Interfaces
- Small Interfaces
- Explicit Interfaces
- Information Hiding

Aufgabe 1: Grundlagen

d) Nennen und erläutern Sie drei beliebige GRASP Patterns.

- Information Expert
- Creator
- Controller
- Low Coupling
- High Cohesion
- Indirection
- Polymorphism
- Pure Fabrication
- Protected Variations
- Pure Fabrication
- Protected Variations

Aufgabe 2: SantaGiftManager

In der Weihnachtswerkstatt herrscht Hochbetrieb, und die Elfen kommen kaum mit dem Verpacken hinterher. Aber dieses Jahr ist alles anders – dank des SantaGiftManagers! Dieses revolutionäre Tool bringt Ordnung ins Geschenkchaos und sorgt dafür, dass kein Geschenk ohne Schleife bleibt. Egal ob rechteckige Box oder sternförmiges Meisterwerk, der SantaGiftManager organisiert alles mit Stil und Präzision.

Die Weihnachtsmann & Co. KG hat Großes vor. Die Elfen planen eine Erweiterung des Tools, um *mehrere Geschenke gleichzeitig zu verwalten* und für diese *automatisierte Routen zu planen*. Erstmal wird sich auf Kuscheltiere, wie Bären, Hasen oder Katzen konzentriert, für die in einem Arbeitsschritt Lieferanweisungen kinderleicht organisiert werden. Und das ist erst der Anfang! Die Elfen haben angedeutet, dass bald auch neue Geschenktypen hinzukommen sollen. Bleibt gespannt – Weihnachten wird nie wieder so chaotisch wie früher!

Für den Anfang haben sich die Elfen einen Prototypen überlegt, der dafür genutzt werden soll, die Kuscheltiere automatisiert zu versenden. Da sie aber nicht allzu bewandert mit der Programmiertechnik sind, haben sie Sie hinzugezogen, um den Prototypen zu bewerten und Tipps zu erhalten, was sie anpassen könnten und sollten.

Aufgabe 2: SOLID-Prinzipien

Welche Solid-Prinzipien gibt es?

- Single-Responsibility Principle: Jede Software Entität soll Verantwortung über eine bestimmte Teilfunktionalität haben
- Open-Closed Principle: Software Entitäten sollten offen für Erweiterungen, aber geschlossen für Modifikationen sein
- Liskov Substitution Principle: Jedes Objekt sollte anstelle der Elternklasse (falls vorhanden) verwendet werden können, ohne dass sich das Verhalten des Programmes ändert
- Interface Segregation Principle: Kein Klient sollte dazu gezwungen werden, abhängig von Methoden zu sein, die er nicht nutzt
- Dependency Inversion Principle: Module höherer Ebenen sollten nicht von Modulen unterer Ebenen abhängig sein, stattdessen nur von Abstraktionen
  - (nach Uncle Bob Martin:) DIP ist gegeben, wenn OCP und LSP konsequent angewendet werden
  - Bei DIP geht es um explizite Abhängigkeiten konkreter Module, OCP kann deutlich mehr beschreiben

Aufgabe 2: Anforderungsbeschreibung

a) Gegen welches SOLID-Prinzip verstößt der gegebene Prototyp? Begründen Sie Ihre Antwort.

b) Passen Sie den Prototypen dem SOLID-Prinzip entsprechend an. Hierfür können Sie entweder (a) Quelltext oder (b) eine genaue Beschreibung der Änderungen, die am Prototypen vorgenommen werden müssen, angeben.

```java
public class SantaGiftManager {
    private RoutePlanner routePlanner;
    private List<PlushRabbit> rabbitGifts;
    private List<PlushBear> bearGifts;
    private List<PlushCat> catGifts;

    public void addAddress() {
        for (PlushRabbit rabbit: rabbitGifts) {
            rabbit.addAddress(routePlanner);
        }
        for (PlushBear bear: bearGifts) {
            bear.addAddress(routePlanner);
        }
        for (PlushCat cat: catGifts) {
            cat.addAddress(routePlanner);
        }
    }
}

public class PlushRabbit {
    private Address address;

    public void addAddress(RoutePlanner routePlanner) {
        routePlanner.addStop(address, <mehr Geschenkinformationen>)

    }
}

public class PlushBear {
    private Address address;

    public void addAddress(RoutePlanner routePlanner) {
        routePlanner.addStop(address, <mehr Geschenkinformationen>)

    }
}

public class PlushCat {
    private Address address;

    public void addAddress(RoutePlanner routePlanner) {
        routePlanner.addStop(address, <mehr Geschenkinformationen>)

    }
}
```

Aufgabe 2: Anforderungsbeschreibung

Die Elfen haben Ihren Anmerkungen konzentriert zugehört und einen neuen Prototyp entworfen. Sie bitten Sie darum, diesen genauso zu bewerten, wie den ersten.

a) Gegen welches SOLID-Prinzip verstößt der gegebene Prototyp? Begründen Sie Ihre Antwort.

b) Passen Sie den Prototypen dem SOLID-Prinzip entsprechend an.

```java
public class SantaGiftManager {
    private RoutePlanner routePlanner;
    private List<PlushGift> plushToys;

    public void addAddress() {
        for (PlushGift toy : plushToys) {
            if (toy instanceof PlushRabbit) {
                Address address = ((PlushRabbit) toy).getAddress();

                this.routePlanner.addStop(address);
                System.out.println("Hase entdeckt!");
            } else if (toy instanceof PlushBear) {
                Address address = ((PlushBear) toy).getAddress();
                this.routePlanner.addStop(address);
                System.out.println("Bär entdeckt!");
            } else if (toy instanceof PlushCat) {
                Address address = ((PlushCat) toy).getAddress();
                this.routePlanner.addStop(address);
                System.out.println("Katze entdeckt!");
            }
        }
    }
}

abstract class PlushGift {
    private Address address;

    public Address getAddress() {
        return this.address;
    }
}


public class PlushRabbit extends PlushGift {
    @Override
    public Address getAddress() {
        return super.getAddress();
    }
}

public class PlushBear extends PlushGift {
    @Override
    public Address getAddress() {...}
}

public class PlushCat extends PlushGift {
    @Override
    public Address getAddress() {...}
}
```

UNIVERSITÄT LEIPZIG

Fragen?

B.Sc. Annemarie Wittig
annemarie.wittig@informatik.uni-leipzig.de

Nächste Übung: 08.01.2025 (Regulärer Übungsinhalt!)

Den Elfen sind das doch zu viele Regeln. Sie sind doch nicht mehr so überzeugt davon, den SantaGiftManager nochmal vor der Weihnachtssaison anzupassen. Vielleicht reicht Stift und Papier noch ein weiteres Jahr aus…

In dem Sinne ein **Frohes Fest** denen, die feiern und ansonsten eine **schöne Vorlesungsfreie Zeit**!