Abgabe: Aufgabe 7 ST WS 14/15
HS Mannheim – Studiengang Informatik
Sebastian Schuler (1332256)
Thomas Kämmerling (1320565)

Aufgabe 7: Entwurf 3

Fragen

7.2.1)

Lesen Sie die Beschreibungen zu den Entwurfsmustern „Beobachter“ und „Fassade“. Können Sie sich vorstellen, eins dieser Design-Pattern in Ihr bisheriges Design der Studenten-Termin-Applikation einzubauen (oder benutzen Sie dort schon eins)? Begründen Sie:

- ***Falls Sie keins benutzen möchten: Warum nicht?***
- ***Falls Sie eins benutzen (möchten): Welchen konkreten Nutzen erwarten Sie davon in Ihrem Programm? (Bitte keine Allgemeinplätze wie „das Pattern verbessert die Wartbarkeit“!)***

Wir würden hier am ehesten auf das Beobachter-Pattern zurückgreifen. Dies erleichtert es, die App zu aktualisieren, wenn Termine an irgendeiner Stelle geändert werden. So kann ein Beobachter die verschiedenen Qullen für die Termine beobachten und bei einer Änderung kann die App dann eine Aktion ausführen (z.B. die Anzeige aktualisieren). Damit kann man beliebige weitere Datenquellen anbinden und so eine hohe Flexibilität wahren.

Die Nutzung eines Patterns sollte jedoch im ganz konkreten Fall noch einmal geprüft werden. Es ist abzuwägen, ob der Aufwand den Nutzen rechtfertigt. Bei der eher kleinen und in der Nutzung sehr spezifischen App ist dies eventuell zu viel des Guten.

7.2.2)

Was fanden Sie schwierig, oder haben Sie nicht verstanden? Es mag sein, dass Sie hier nichts angeben können. Dann antworten Sie bitte mit „nichts“.

Nichts.

7.2.3)

Beschreiben Sie, was Sie am interessantesten oder gewinnbringend fanden.

Das Kennenlernen der beiden Pattern und das querlesen bei anderen Patterns ist sehr spannend und sicherlich in der Zukunft hilfreich.

Abgabe: Aufgabe 7 ST WS 14/15
HS Mannheim – Studiengang Informatik
Sebastian Schuler (1332256)
Thomas Kämmerling (1320565)

7.2.4)

Welche Anknüpfungspunkte sehen Sie zwischen diesem Stoff und dem, was Sie bereits wissen?

Verschiedene Pattern wurden bereits in OOT angesprochen, hier kann das Wissen nun vertieft werden.

7.2.5)

Wie lange haben Sie gelesen, Fragen beantwortet, Aufgaben bearbeitet? Gefragt ist jeweils der Gesamtaufwand aller Gruppenmitglieder.

4 Stunden

Aufgaben

7.3.1)

Überlegen Sie sich zu jedem der fünf SOLID-Prinzipien ein (kleines) Beispiel und beschreiben Sie die konkreten Auswirkungen des Prinzips. Falls möglich, wählen Sie Ihre Beispiele im Kontext der Studenten-Termin-Applikation; Sie dürfen die Aufgabenstellung entsprechend anpassen.

Single Responsibility Principle

Es sollte nur einen Grund geben eine Klasse zu ändern.

Die Logik, die die eingehenden Termininformationen aus den verschiedenen Datenquellen verarbeitet und die Darstellung sollten getrennt werden. Es muss dafür also zwei Komponenten geben. In der Darstellung dürfen nur „fertige“ Ergebnisse aufbereitet werden. In der Logik werden die eingehenden Daten sortiert, gesammelt und verarbeitet. Damit muss bei Änderungen an der Logik die Darstellung nicht angefasst werden und umgekehrt.

Open Closed Principle

Software Komponenten (Klassen, Module, Funktionen, etc.) sollen offen für Erweiterungen sein, aber geschlossen für Modifikationen.

Es ist in der Termin-App möglich Termine von verschiedenen Systemen abzurufen (POS und Terminverwaltung). Falls einige bzw. neue Termine in Zukunft mit anderen Systemen verwaltet werden, muss es möglich sein auch dieses System in die App einzubinden und von ihr Informationen anzufordern. Allerdings darf es nicht möglich sein die App so zu modifizieren, dass die angeforderten Informationen nicht mehr gespeichert werden können. Diese Erweiterbarkeit kann durch eine Quellverwaltungskomponente ermöglicht werden, von der konkrete Komponenten für die Terminabfrage abgeleitet sind

Liskov Substitution Principle

Funktionen die Referenzen benutzen um Klassen zu gründen sollten fähig sein Objekte von abgeleiteten Klassen zu verwenden ohne diese zu kennen.

Die Logik sollte es nicht kümmern, welche der Terminquellen verwendet wird. Die Logik sollte einfach ein „Terminobjekt“ bekommen und damit dann weiterarbeiten. Es darf nicht jedes Mal nötig sein zu wissen, woher die Daten genau kommen. Auch dies kann über eine übergeordnete „Quellenverwaltung“ realisiert werden. (Mittels Vererbung, Abstraktion oder welche Möglichkeiten die gewählte Sprache auch immer bietet.)

Interface Segregation Principle

Nutzer sollten nicht von Interfaces abhängig gemacht werden, die sie nicht benutzen.

Ein Interface soll nur genau das implementieren bzw. vorgeben, was auch jede „Unterklasse“ benötigt. So sollte eine Quellverwaltungskomponente ihren Kindern keine Loginfunktion mitgeben, da diese nur im POS benötigt wird. Diese Funktion ist dann direkt in der „POS-Komponente“ zu implementieren.

Dependency Inversion Principle

A. High-Level Einheiten sollten nicht von Low-Level Einheiten abhängig sein. Beide sollten von Abstraktionen abhängig sein.

Es gibt eine Klasse die Informationen vom POS und eine die Informationen von der HS-Terminverwaltung anfordert. Diese sollten nicht voneinander abhängig bzw. voneinander abgeleitet sein, sondern beide von einer abstrakten Basisklasse abgeleitet werden. Somit ist eine einwandfreie Funktionsweise eher gewährleistet.

B. Abstraktionen sollten nicht von Details abhängig sein. Sondern anders herum.

Dies klingt für uns ähnlich dem Interface Segregation Principle. Sprich nur weil zum Beispiel die POS-Komponente ein Login benötigt sollte dies dadurch nicht automatisch für die Abstraktion und damit für andere Abfragekomponenten dienen.

Abgabe: Aufgabe 7 ST WS 14/15
HS Mannheim – Studiengang Informatik
Sebastian Schuler (1332256)
Thomas Kämmerling (1320565)

7.3.2)

Diskutieren Sie die Studenten-Termin-Applikation im Hinblick auf jede der folgenden vier Faktorengruppen aus „Balzert: Einflussfaktoren auf die Architektur“:

1. ***Anwendungsart***
2. ***Verteilungsart***
3. ***Kontext des Anwendungssystem***
4. ***Art der softwaretechnischen Infrastruktur***

Welche Entscheidungen treffen Sie jeweils für Ihre Applikationsarchitektur und warum diese bzw. warum keine andere?

Anwendungsart

Die Studenten-Termin-Applikation ist eindeutig eine *Einzelplatz-Anwendung*, da sie auf einem Gerät läuft und sich mehrere Studenten nacheinander einloggen und ihre individuellen Termine abrufen können. Da sie eine unabhängige App ist, ist sie eine *Stand-Alone-Anwendung*. Weiterhin ist sie unserer Meinung nach auch eine teils *Stapelverarbeitungs-Anwendung* da sie automatisch nach dem einloggen die bereits gespeicherten Termine anzeigt und sie nun aktualisiert.

Verteilungsart

Hier kommt nach unserer Meinung nur eine *Serviceorientierte Architektur* in Frage, da die Applikation weder permanent in Verbindung mit dem Server steht, noch über den Standard-Webbrowser funktioniert oder gleichberechtigte Computersysteme miteinander kommunizieren.

Kontext des Anwendungssystem

Die App greift auf verschiedene Datenquellen der Hochschule zu, die zur Aktualisierung zur Verfügung stehen müssen. Eine Nutzung ist auch ohne eine Erreichbarkeit der Quellen eingeschränkt möglich, da es eine Persistenzkomponente gibt.

Art der softwaretechnischen Infrastruktur

Da auf bereits existierende Lösungen (POS und HS-Terminverwaltung) zugegriffen werden muss und sich diese von der Architektur her mit der App unterscheiden wird eine Heterogene Laufzeit- und Entwicklungsumgebung gewählt.