Praktikumsanleitung – Versuch 6

Architekturentwurf

Praktikumstermine: Mo, 10.05.2004 / Di, 11.05.2004 / Do, 06.05.2004

Einführung

Das Erstellen eines Entwurfs teilt sich auf in die Konzipierung und Erstellung eines Architekturentwurfs und in die eines Detailentwurfs. Der Architekturentwurf (auch Grobentwurf genannt) hat zum Ziel, ein Lösungskonzept eines zu erstellenden Systems zu erarbeiten. Schwerpunkte bei der Lösungskonzeption sind die Gliederung der Software in Module, die Klärung von Nebenläufigkeiten und die Ressourcenzuteilung. Die Versuche 6 und 7 haben zum Ziel, für die Anforderungsspezifikation «Börsencafé» einen objektorientierten Entwurf zu erstellen. Aspekte wie Datenschutz, Fehlertoleranz, u.ä. sollen hierbei vernachlässigt werden.

In diesem Versuch wird im Rahmen eines Architekturentwurfs das Projekt modularisiert, die Vererbungshierarchie für das gegebene Projekt wird entwickelt und Klassenbeschreibungen werden erstellt. Anschliessend beginnen Sie, die Klassen des Architekturentwurfs im Detailentwurf zu präzisieren und in entsprechende Java-Konstrukte umzusetzen. Ziel des Detailentwurfs ist ein dokumentiertes Programmskelett (Klassendeklarationen und Methoden-Beschreibungen), welches Vorgabe für die eigentliche Implementierung ist. Der Architekturentwurf erfolgt für das Gesamtsystem; ein Detailentwurf ist nur für einen Teil des Systems zu erstellen.

In Versuch 7 werden die Ergebnisse des Architekturentwurfs einem technischen Review unterzogen und der Detailentwurf für das Teilsystem wird fertig ausgearbeitet.

Vorbereitung des Versuchs (Heimarbeit)

Obligate Einarbeitung in die Thematik. [Glin98] gibt in Kapitel 6 eine Einführung in das Erstellen des Architekturentwurfs, in [Glin96] in Kapitel 5 werden Objekt- und Klassenmodelle beschrieben. [Somm01] führt in Kapitel 10 kurz in den Entwurf von Software ein, der Design-Prozess und verschiedene Design-Strategien werden vorgestellt und schliesslich werden bestimmte Qualitätsmerkmale eines Entwurfs kurz angesprochen. Kapitel 11 im selben Buch führt kurz in den objektorientierten Entwurf ein. Zur Vertiefung kann [Garl96] (als Einführung in

die Architektur von Software) oder auch [Free83] (Überblick über verschiedene Entwurfstechniken anhand von Originalartikeln) gelesen werden.

Durchführung des Versuchs

Aufgabe 6.1: Modularisierung

Auf Grundlage des in Aufgabe 4 entwickelten und in Versuch 5 geprüften objektorientierten Anforderungsmodells soll eine Klassenbeziehungsstruktur/Benutzungsstruktur entwickelt werden (bezüglich Modularisierung siehe [Glin98], Kapitel 6). Bei der Entwicklung der Benutzungsstruktur sind folgende Aspekte zu klären:

- In welche Klassen wird das System zerlegt? (Hinweis: Neben den im Anforderungsmodell aufgeführten Klassen sind unter Umständen weitere Klassen hinzuzufügen bzw. bestehende Klassen zu ersetzen, siehe nächster Abschnitt).
- Wie hängen die Klassen untereinander zusammen (welche Benutzungszusammenhänge, bzw. Beziehungen bestehen zwischen den Klassen)?
- Wie integriert sich die Anwendungs- bzw. Applikationslogik in die zur Verfügung gestellten Klassen (siehe auch Aufgabe 6.2).

Die Klassen der Modularisierung ergeben sich aus den Klassen, die in der Anforderungsspezifikation gefunden und spezifiziert wurden. Diese «Anforderungs»-Klassen sind zusätzlich durch «Informatik»-Klassen zu ergänzen, die für die Realisierung nötig sind (z.B. diverse Manager-, Verwalter-, Darstellungs-, Controller-, Ressourcen-Klassen). Darüber hinaus kann es notwendig sein, die in der Spezifikation ermittelte Struktur zu reorganisieren und in eine azyklische Benutzungsstruktur zu überführen.

Verwenden Sie zur Modellierung der Benutzungsstruktur Together: Zur Zusammenfassung von Klassen kann das Konstrukt *Packages* verwendet werden, Benutzungsbeziehungen werden in UML mittels *Dependencies* modelliert.

Aufgabe 6.2: Vererbungshierarchie

Entwickeln Sie für die im Teilversuch 6.1 bestimmten Klassen eine Vererbungshierarchie. Orientieren Sie sich bei der Modellierung der Vererbungshierarchie an den im Anforderungsmodell aufgeführten Spezialisierungsstrukturen. Stellen Sie die Einbindung der applikationsspezifischen Klassen in die Klassenbibliotheken von Java dar.

Hinweis: Die im Anforderungsmodell definierten Spezialisierungsstrukturen müssen sich nicht zwingend im Entwurfsmodell wiederfinden. Unter Umständen ist es sinnvoll, Spezialisierungen nicht durch Vererbungsstrukturen zu realisieren. Andererseits können im Entwurf

Vererbungsstrukturen hinzugefügt werden, um Komponenten einfacher wiederzuverwenden oder kompaktere Modelle zu erhalten.

Stellen Sie Aufgabe 1 und 2 in einem UML-Klassendiagramm dar.

Aufgabe 6.3: Coderahmen/Detailentwurf

Beginnen Sie den Detailentwurf derjenigen Klassen, welche die Anwendungsfälle «(variable) Preise berechnen», «Anzeige der Speisen, Getränke und Kurse an der Anzeigetafel» und «Angebote verwalten: Speisen/Getränke einfügen, ändern, löschen» realisieren. Vergessen Sie die Applikation als solches nicht.

Dieser Coderahmen enthält für jede Klasse Definitionen aller benötigter Attribute (Klassen- und Instanzvariablen) und die Köpfe der Klassen- und Instanzmethoden. Die Methoden müssen noch keinerlei Implementierung enthalten. Dokumentieren Sie die Java-Klassen den Richtlinien zur Java-Entwicklung [Bern98] entsprechend durch:

- Allgemeine Angaben zur Klasse (wie etwa Autoren, Projekt, Version, Zweck, Verantwortlichkeit, usw.). Siehe hierzu die Entwicklungsrichtlinien Kapitel 5.3.
- Deklaration der Methoden. Hier wird besonders Wert auf die Deklaration der Parameterschnittstelle und des Kopfkommentars einer jeden Methode gelegt (siehe Entwicklungsrichtlinien 5.5).
- Atribute (kurze Beschreibung). Siehe hierzu Entwicklungsrichtlinien Kapitel 5.8.

Die Modellierung des Detailentwurfs erfolgt in Eclipse.

Die Signatur der Methode zur Berechnung der variablen Preise ist innerhalb der Berechnungsklasse **vorgegeben** und darf **nicht** verändert werden. -> **siehe Ende dieses Dokuments**

Am Ende des Versuchstermins(!) werden die Entwurfsdokumente, die in den Teilaufgaben 6.1 bis 6.3 erstellt wurden, das zugehörige Anforderungsmodell (Klassen- und Verhaltensmodell), auf dem der Entwurf aufbaut, und die entsprechenden Anwendungsfälle (aus Versuch 1 bis 3) an die Betreuer (OLAT) weitergegeben.

Hinweis

- Es ist sinnvoll, den groben Architekturentwurf (welche Module, bzw. Klassen benötigt man, welche Benutzungsbeziehungen gelten) im Team zu erarbeiten und bei der präzisen Architekturbeschreibung (Verantwortlichkeit und Zweck einer Klasse...) die Arbeit aufzuteilen. Beim Detailentwurf (Beschreibung von Datenstrukturen, Methodenbeschreibung...) ist ebenfalls eine Arbeitsaufteilung sinnvoll.

- Die Teilversuche 6.1, 6.2 und 6.3 sind nicht zwingend in dieser Reihenfolge durchzuführen, vielmehr ist es sinnvoll und zweckmässig, die Dokumente, insbesondere die Modularisierung und die Vererbungshierarchie, parallel zu entwickeln (wechselseitige Abhängigkeiten).
- Den Architektur- und Detailentwurf parallel zu entwickeln, ist mit einigen Risiken behaftet (der Detailentwurf wird aufgrund eines noch nicht geprüften und nicht abgeschlossenen Architekturentwurfs gemacht und kann damit a priori fehlerhaft sein), dieses Vorgehen ist aber in der Praxis recht häufig, da entsprechend Zeit gewonnen werden kann.

Abgabe

- Klassendiagramm aus Aufgabe 6.1 und 6.2
- Neueste Version der Anwendungsfälle aus Versuch 3
- In Versuch 5 geprüfte und überarbeitete Version von Klassen- und Verhaltensmodell der Anforderungen

Die Abgabe erfolgt am Ende des Versuchstermins im OLAT Gruppenordner in einem von Ihnen anzulegenden Verzeichnis mit der Bezeichnung „Versuch06_Lsg“.

- Coderahmen der Klassen aus Aufgabe 6.3

Die Abgabe erfolgt 3 Tage nach dem Versuch im gleichen OLAT Ordner.

Abgabetermin

- Klasse Montag: Mo, 10.05.04 18:00 (Aufg. 6.1 und 6.2); Do 13.05.04 24:00 (Aufg. 6.3)
- Klasse Dienstag: Di, 11.05.04 18:00 (Aufg. 6.1 und 6.2); Fr 14.05.04 24:00 (Aufg. 6.3)
- Klasse Donnerstag: Do, 06.05.04 18:00 (Aufg. 6.1 und 6.2); So 09.05.04 24:00 (Aufg. 6.3)

Unterlagen in OLAT:

- [Bern98] S. Berner, M. Glinz, S. Joos, N. Schett (1998): *Entwicklungsrichtlinien für Java-Software - Version 3.0.1*. Institut für Informatik der Universität Zürich, 1998. (aus Versuch 2)
- Intro UML (aus Versuch 2)

Referenzen:

- [Garl96] D. Garlan, M. Shaw: *An Introduction to Software Architecture*. In: V. Ambriola, G. Tortora: *Advances in Software Engineering and Knowledge Engineering, Vol. 1*. New Jersey: World Scientific Publishing Company, 1993.
- [Glin98] M. Glinz: Vorlesungsskript WS 98/99, *Software Engineering I*. Institut für Informatik der Universität Zürich, Kapitel 6.
- [Glin96] M. Glinz: Vorlesungsskript SS 96, *Informatik, Teil 2, Modellierung*. Institut für Informatik der Universität Zürich, Kapitel 5.
- [Somm01] I. Sommerville: *Software Engineering*. Fourth Edition, Wokingham: Addison-Wesley, 1992. (- -> IfI-Bibliothek, D.2 8359 und D.2 9200)

- [Free83] P. Freeman, A.I. Wasserman: *Tutorial on software design techniques*. IEEE Computer Society Press Silver Spring, Md., 1983. (--> HBI)

Signatur Preisberechnungsklasse

```
import java.util.Calendar;

/**
 * This class contains methods and attributes needed to
 * calculate the current price of variable priced articles.
 * The price is calculated on each order.
 *
 * @author      Team Sopra
 * @copyright   Department for Computer Science, Requirement
                Engineering Group
 * @history     2004-24-03 RG First Version
 * @version     2004-24-03 RG 1.0
 * @responsibilities  This class calculates prices for variable
 *                    priced articles.
 * /
public class PriceCalculator {

/**
* Represents the last time the price was calculated (time of the
* last order).
*/
private Calendar lastOrderTimeStamp;


/**
 * This method calculates the price of variable price articles
 * depending of the amount sold since the last calculation of
 * the price, the expected standard amount of sales during
 * the interval between two calculations, the standard and
 * current prices and the upper and lower bounds of the price.
 *
 * @pre    0.0 < lowerBoundary < upperBoundary AND
 *         amountSold >= 0 AND
 *         standardAmountSold >= 0
 * @post   lowerBoundary <= return <= upperBoundary
 * @invariant lowerBoundary@POST == lowerBoundary@PRE
 * @invariant upperBoundary@POST == upperBoundary@PRE
 * @invariant standardPrice@POST == standardPrice@PRE
 * @invariant standardAmountSold@POST == standardAmountSold@PRE
 * @param  standardPrice: a float representation of the base
 *              price of an article
```

```
 * @param  currentPrice: a float representation of the current
 *               selling price of an article
 * @param  lowerBoundary: a float representation of the lowest
 *               possible price of an article
 * @param  upperBoundary: a float representation of the highest
 *               possible price of an article
 * @param  amountSold: an int representation of the amount sold
 *               since the last calculation of the price
 * @param  standardAmountPerHour: an float representation of the
 *               expected sales during one hour
 * @param  timeStamp: a Calendar representation of the time the
 *               article was ordered
 * @return a float representation of the calculated new price
 */
public float calculatePrice(     float standardPrice,
                                 float currentPrice,
                                 float lowerBoundary,
                                 float upperBoundary,
                                 int amountSold,
                                 float standardAmountPerHour,
                                 Calendar timeStamp) {

     //implementation…

     }
}
```