

PM2, Aufgabenblatt 3

Programmieren 2 – Sommersemester 2017

Modularisierung, GUI-Programmierung, Fachwerte

Ausgabedatum: 15. Mai 2017

Kernbegriffe

Module sind klassisch statische Einheiten des Programmtextes mit einer Schnittstelle und einer (gekapselten) Implementation. Ein Modul deklariert über seine *Import-Schnittstelle* (auch outgoing interface) die Einheiten eines Softwaresystems, die es zur Erfüllung seiner Aufgabe benötigt. Seine eigenen Dienstleistungen bietet es über seine *Export-Schnittstelle* (auch incoming interface) an. Wenn Module in Modulen geschachtelt sein können, dann beeinflusst dies die *Benennung* von Modulen und den *Zugriffsschutz* auf geschachtelte Einheiten. In Java sind die Klassen lediglich modulähnlich: sie können ineinander geschachtelt werden, ihre Implementation verstecken und haben eine implizite Export-Schnittstelle, aber keine explizit deklarierte Import-Schnittstelle. Die *Pakete* (engl.: packages) in Java sind ebenfalls nur modulähnlich: Ihre Schachtelung beeinflusst lediglich ihre Benennung, nicht aber ihren Zugriffsschutz.

Grafische Benutzungsschnittstellen (engl.: graphical user interfaces, GUIs) sind heutzutage der Standard für interaktive Softwaresysteme. Typische Elemente einer GUI sind u.a. Schaltflächen (engl.: buttons), Beschriftungen (engl.: labels), Listen und Editoren, die in Fenstern (engl.: windows) angeordnet sind. Drei Fragen müssen für den technischen Entwurf einer GUI beantwortet werden: Welche GUI-Elemente werden verwendet? Wie sind diese angeordnet und wie reagieren sie auf Größenänderungen des Fensters (falls zugelassen)? Wie wird in der Software auf Ereignisse an der Oberfläche reagiert?


Klassisch definieren *Objekttypen* in der Objektorientierung *Objekte*, die eine Lebensdauer haben (erzeugt und vernichtet werden) und einen veränderlichen Zustand haben können. Die Operationen auf diesen Objekten können den Zustand des Objektes selbst und/oder beliebige andere Objekte verändern. *Werttypen* hingegen definieren *Werte* samt den auf ihnen verfügbaren Operationen. Werte sind verglichen mit Objekten stark eingeschränkt: Werte werden nicht erzeugt oder vernichtet und haben keinen veränderlichen Zustand. Werttypen definieren ausschließlich Operationen, die keine Zustände verändern (*seiteneffektfrei* sind) und für dieselben Parameter immer dasselbe Ergebnis liefern (*referentiell transparent* sind). In Java sind die *primitiven Typen* Werttypen, aber auch der Typ *String* kann als Werttyp gesehen werden. Java bietet viel Unterstützung bei der Definition benutzerdefinierter Objekttypen, aber nur sehr wenig bei der Definition benutzerdefinierter Werttypen. Deshalb müssen Programmierer Werttypen mit *Wertklassen* modellieren, bei denen die Entwickler die Eigenschaften von Werttypen selbst per Konvention und mit Disziplin einhalten müssen. Werttypen, die *fachliche Konzepte* (wie Kontonummern, Postleitzahlen, Geldbeträge) direkt abbilden, nennen wir *Fachwerttypen*.

Aufgabe 3.1 Pakete in Eclipse

Java bietet die Möglichkeit, Software zu modularisieren, indem Klassen auf Pakete (engl: packages) verteilt werden. Eure Mediathek ist noch nicht in Pakete aufgeteilt, während das Kinoticketssystem, das ihr in der nächsten Aufgabe erweitern sollt, schon eine Paketstruktur hat, die sich nach den PM2-Entwurfsregeln richtet. Das Kinosystem liegt im Projekt *Kinoticketverkauf_VorlageBlatt03* vor.

Ihr sollt nun der Mediathek die letzte Ehre erweisen und ihren Quelltext in gleicher Weise auf Pakete aufteilen. Eclipse unterstützt bei solchen *Refactorings* (Änderungen an der Struktur des Quelltextes, die für den Benutzer der ausgeführten Software nicht sichtbar werden).

3.1.1 Als erstes sollt ihr alle Klassen und Interfaces (also alle Typen) aus dem momentan namenlosen Paket in ein neues Paket bewegen. Öffnet dafür auf dem src-Verzeichnis im Package-Explorer das Kontextmenü. Selektiert *New->Package* und erzeugt ein Paket namens *mediathek*. Nun könnt ihr alle Typen in das neue Paket verschieben, indem ihr alle selektiert und dann im Kontextmenü *Refactor->Move...* auswählt.

 3.1.2 Verschiebt nun alle Medien-Typen in ein eigenes Unterpaket *medien*. Welcher Fehler tritt danach auf? **Schreibt einen kurzen Text, der die Situation erläutert und die Lösungsmöglichkeiten diskutiert.** Behebt das Problem anschließend. Überlegt euch auch, wie mit den zugehörigen Testklassen umzugehen ist.

- 3.1.3 Alle Klassen und Interfaces sollt ihr nun anhand ihrer fachlichen und technischen Ausprägung in passende Pakete gemäß den PM2-Entwurfsregeln verschieben:
- Neben Medien gibt es noch weitere Klassen, die Dinge oder Personen aus dem Anwendungskontext abbilden. Welches Paket bietet sich an?
 - Es gibt fachliche Dinge, die einen Wertcharakter besitzen, und deshalb durch Fachwertklassen modelliert wurden. Ein Beispiel hierfür ist `PLZ`. Welches Paket bietet sich an?
 - Einige Klassen und Interfaces verwalten Materialien, bieten systemweite fachliche Funktionen an oder die Persistierung von Daten. Welches Paket bietet sich an?
 - Andere Klassen bilden die Benutzungsschnittstelle ab, hierzu zählen `AusleihWerkzeug` und `AusleihUI`. Welches Paket bietet sich an?
 - Bei allen übriggebliebenen Klassen und Interfaces müsst ihr nun nachschauen, wer von Ihnen erbt oder wer sie verwendet, um eine sinnvolle Einordnung in unser Schema zu finden.
- 3.1.4 In Java existiert eine Konvention für Paketnamen: Als Präfix verwendet man eine umgedrehte URL, um die Namen möglichst eindeutig zu machen. Anstatt des einfachen Präfixes *mediathek* kann beispielsweise der Präfix *de.hawhh.informatik.sml.mediathek* verwendet werden. Ändert die Paketnamen entsprechend.
- 3.1.5 *Zusatzaufgabe:* Überlegt euch für jedes Paket, ob es sinnvoll ist, ein oder mehrere untergeordnete Pakete zu ergänzen. Erstellt diese Pakete und verschiebt die passenden Klassen.

Aufgabe 3.2 Ein Werkzeug zur Barzahlung

In dieser Aufgabe sollt ihr das Kinoticketsystem erweitern, mit dem an der Kasse eines Kinos Karten für die Vorstellungen verkauft werden können.

Wenn ihr das System über die Startup-Klasse startet, seht ihr eine weitgehend fertige Anwendung. Es kann zwischen den Tagen geblättert werden, es kann eine Vorstellung ausgewählt werden und es können Plätze eines Saals selektiert und verkauft werden.

Die Nutzer des Kinoticketsystems wünschen eine Unterstützung bei der Abwicklung von Barbezahlungen an der Kinokasse. Ein Interview hat folgende **Anforderungen** für ein Werkzeug ergeben:

- a) *Nach dem Auswählen von Plätzen einer Vorstellung und einem Klick auf „Verkaufen“ erscheint ein neues Fenster zur Barzahlung.*
- b) *In diesem werden der*
 - 1) *aufsummierte Preis für alle ausgewählten Plätze,*
 - 2) *der vom Kunden bezahlte Betrag*
 - 3) *und das mögliche Rückgeld dargestellt.*
- c) *Der vom Kunden bezahlte Betrag wird über die Tastatur eingegeben.*
- d) *Das mögliche Rückgeld wird automatisch berechnet.*
- e) *Das Fenster bietet zwei Schaltflächen:*
 - 1) *„Abbrechen“: Der Verkaufsvorgang wird abgebrochen, die Karten werden nicht verkauft und dementsprechend als frei im Kinoticketsystem angezeigt.*
 - 2) *„OK“: Die Karten werden verkauft und dementsprechend im Kinoticketsystem markiert. Diese Option darf nur auswählbar sein, wenn der Restbetrag ≥ 0 ist.*

In dieser Aufgabe soll dem System ein Werkzeug zur Barzahlung hinzugefügt werden. Ihr sollt diese Aufgabe in **zwei Schritten** bearbeiten: Als erstes sollt ihr eine Lösung für die unten beschriebenen Anforderungen **entwerfen**. Erst in einem explizit zweiten Schritt sollt ihr diesen Entwurf **implementieren**.

- 3.2.1 Zeichnet euch zum Einstieg in das neue System ein **Klassendiagramm** der Materialien. Es sollte deutlich werden, welche Materialien welche anderen verwenden. Erweitert dieses Diagramm in

einem weiteren Schritt um die Fachwerte. Fügt erst dann die Werkzeuge dem Diagramm hinzu.
Bringt dieses Diagramm zur Abnahme auf Papier mit.

3.2.2 Wie bereits gesagt, folgt auch das Kinoticketsystem den PM2-Entwurfsregeln. Welchen Unterschied zwischen den Systemen seht ihr, wenn ihr euch nur an der internen Struktur orientiert?

3.2.3 Das Entwerfen eines Softwarewerkzeugs betrifft zwei Ebenen. Zum einen muss eine geeignete Benutzungsschnittstelle entwickelt werden, die die Bearbeitung der Aufgabe durch einen Nutzer sinnvoll unterstützt und die technischen Rahmenbedingungen des UI-Frameworks beachtet. Zum anderen muss das Werkzeug mit der Programmlogik verknüpft werden.

Erarbeitet einen Lösungsvorschlag für die von den Nutzern gestellten Anforderungen, der beide Ebenen adressiert. Euer Vorschlag soll die folgenden Fragen beantworten:

- Wie sollte eine *grafische Benutzungsschnittstelle* zur Unterstützung der Barbezahlung an der Kinokasse gestaltet sein? **Macht euch eine Skizze!**
- Welche Widgets und Layoutmanager von Java Swing könnt ihr in eurem UI-Entwurf nutzen? **Kennzeichnet diese in eurer Skizze.** Zusätzlich zum Skript findet ihr Informationen hier:
<http://docs.oracle.com/javase/tutorial/uiswing/components/index.html>
<http://docs.oracle.com/javase/tutorial/uiswing/layout/index.html>
- Aus welchen Klassen bestehen Werkzeuge gemäß unserer PM2-Entwurfsregeln und welche Aufgaben übernehmen sie?
- Mit welchen Klassen und Interfaces muss euer neues Werkzeug in Beziehung stehen? **Ergänzt das Werkzeug und seine Beziehungen in eurem UML-Klassendiagramm.**
- Welche Materialien benötigt euer neues Werkzeug?
- Was geschieht im Programm,
 - wenn „verkaufen“ geklickt wird und mehrere Plätze selektiert sind,
 - wenn ein Geldbetrag über die Tastatur eingegeben wird und der Restbetrag berechnet wird,
 - wenn auf „Abbrechen“ geklickt wird,
 - wenn auf „OK“ geklickt wird?
- Soll es während des Barzahlungsvorgangs möglich sein, weiterhin mit der restlichen Benutzungsschnittstelle zu interagieren? Welche Konsequenzen hätte dies? Was ist ein **modaler Dialog**? **Schriftlich!**
- Wie werden Geldbeträge im System momentan repräsentiert?

3.2.4 Implementiert euren Lösungsvorschlag erst, wenn ihr **3.2.1 bis 3.2.3 gründlich bearbeitet habt.**

Laut PM2-Entwurfsregeln müsst ihr für Werkzeug- und UI-Klassen keine JUnit-Tests schreiben. Stattdessen solltet ihr euch aber systematisch Gedanken über **Testfälle für die grafische Oberfläche** machen. **Haltet eure Testfälle schriftlich fest** und führt diese Tests bei der Abnahme vor.

Aufgabe 3.3: Geldbeträge als Fachwerte modellieren und implementieren

Das Kinoticketsystem zeigt für alle ausgewählten und verkaufbaren Sitzplätze den Gesamtpreis in Euro-Cent für die Karten an. Alle Berechnungen und Prüfungen werden zurzeit mithilfe des Typs `int` umgesetzt. Eure Aufgabe ist es, dies über einen neuen Fachwerttyp zu gestalten. Für diesen soll gelten:

- a) Er ermöglicht eine String-Repräsentation der Form „EE,CC“.
- b) Er bietet die folgenden Operationen zum Rechnen mit Geldbeträgen an:
 - Addieren und Subtrahieren zweier Geldbeträge
 - Multiplizieren eines Geldbetrags mit einer Zahl
- c) Er bietet Konvertierungshilfen für das Umwandeln von String- und Integer-Objekten bzw. `int`-Werten in Geldbeträge.
- d) Er soll ermöglichen, im Barzahlungswerkzeug kommaseparierte Beträge über die Tastatur einzugeben.

3.3.1 Erarbeitet einen **Lösungsvorschlag**, der die Berechnung und Darstellung von Geldbeträgen anhand eines neuen Fachwerttyps umsetzt. Die **Spezifikation** soll hierbei über eine von euch implementierte **Testklasse** für die zukünftige Fachwert-Klasse erfolgen. Diese Testklasse wird erstmal nicht übersetzbar sein, soll aber bei der Abnahme als Grundlage zur Diskussion dienen. Also **bewahrt diese Vorversion** für die Abnahme auf, denn erfahrungsgemäß müsst ihr die Testklasse anpassen, sobald ihr die Fachwert-Klasse wirklich implementiert.

Diskutiert zudem **schriftlich vorab** (also bevor ihr implementiert) die folgenden Fragen:

- Was ist ein Fachwert und wie ist dieser nach den PM2-Entwurfsregeln aufgebaut?
- Welche Vorteile bietet die Verwendung eines Fachwerttyps gegenüber einem Integertyp?
- Lässt euer Fachwerttyp negative Beträge zu? Erläutert an konkreten Beispielen, welche Konsequenzen eure Entscheidung hat.
- Welche Schnittstelle hat der von euch vorgeschlagene Fachwerttyp? Denkt hierbei auch an das Vertragsmodell.
- An welchen Stellen im Quelltext wird euer Fachwerttyp zukünftig verwendet werden? Markiert diese Stellen durch ToDos und **zeichnet ein für diese Diskussion geeignetes UML-Klassendiagramm**.

3.3.2 Implementiert euren Lösungsvorschlag, so dass zur Berechnung der Ticketpreise **ausschließlich euer Fachwerttyp** verwendet wird. Haltet dabei die vorgegebenen softwaretechnischen Qualitätsmerkmale ein:

- ☐ Tests
- ☐ Vertragsmodell
- ☐ Schnittstellenkommentare
- ☐ Quelltextkonventionen

3.3.3 *Zusatzaufgabe:* Fachwerte können in einem Pool vorgehalten werden, so dass stets dasselbe Exemplar für die Repräsentation eines Werts verwendet wird. Überlegt euch eine softwaretechnische Lösung für dieses Problem und baut diese **in einer Kopie** eures Programms ein.