Karlsruhe Institute of Technology

aifb
Institut für Angewandte Informatik
und Formale Beschreibungsverfahren

Prof. Dr. J.M. Zöllner

Nikolai Polley, Mark Hüneberg
E-Mail: Nikolai.Polley@kit.edu

Programmieren I: Java

Wintersemester 2021/2022

Übungsblatt P08

(Bearbeitung im Rechnerpraktikum in der Zeit vom 17.01.2022 bis zum 21.01.2022)

Dieses P-Übungsblatt behandelt weitere Aspekte der objektorientierten Programmierung sowie Ausnahmebehandlung (Exceptions) und einfach verkettete Listen.

Aufgabe P 8.1 * (Kapselung)

Gegeben sei eine Klasse `Student`. Diese sieht bisher folgendermaßen aus:

```
public class Student {
  private String name; // Name des/der Studierenden
  private int alterInJahren; // Alter des/der Studierenden in Jahren
  private int matrikelNr; // Matrikel-Nummer des/der Studierenden
}
```

Wie Sie sehen können, ist den drei Instanzvariablen der Zugriffsmodifikator `private` vorangestellt, d.h. sie sind nur innerhalb dieser Klasse direkt nutzbar.

a) Damit Sie die Attribute der Klasse nutzen können, müssen Sie diese nach außen hin verfügbar machen. Implementieren Sie dazu für die drei Instanzvariablen geeignete Getter- und Setter-Methoden.

b) Implementieren Sie eine Klasse `Studiverwaltung`, die in ihrer `main`-Methode ein neues `Student`-Objekt erzeugt, dessen Attribute mit geeigneten Werten belegt, diese Werte anschließend wieder ausliest und sie auf der Konsole ausgibt.

Aufgabe P 8.2 */** (Exceptions)

Schreiben Sie eine Klasse `ExceptionFangen`, die nur aus einer `main`-Methode besteht. In dieser soll zunächst ein `String`-Feld der Länge 4 mit den Werten "34", "252", "drei" und "3.5" angelegt werden. Danach soll der Benutzer/die Benutzerin *fünfmal* nach einer Indexposition gefragt werden, woraufhin versucht wird, den Wert an dieser Position mit der `parseInt()`-Methode der Klasse `Integer` in ein Integer-Objekt umzuwandeln und dann auszugeben.

a) Demonstrieren Sie Ihr Programm manuell mit zulässigen und unzulässigen Indexpositionen. Was passiert bei unzulässigen Indexpositionen oder Einträgen, die nicht geparset werden können?

b) Ändern Sie nun Ihr Programm wie folgt ab: Fangen Sie an geeigneter Stelle die mögliche `ArrayIndexOutOfBoundsException` und `NumberFormatExceptions` ab und geben Sie dabei eine entsprechende Fehlermeldung auf der Konsole aus.

c) Demonstrieren Sie Ihr Programm erneut analog zu **a)**:

Ein möglicher Programmablauf sollte wie folgt aussehen:

```
Welche Indexposition soll ausgegeben werden?
4
Sie haben eine unzulaessige Indexposition angegeben.
Welche Indexposition soll ausgegeben werden?
3
Den Wert an dieser Stelle kann man nicht parsen!
Welche Indexposition soll ausgegeben werden?
2
Den Wert an dieser Stelle kann man nicht parsen!
Welche Indexposition soll ausgegeben werden?
1
feld[1] = 252
Welche Indexposition soll ausgegeben werden?
0
feld[0] = 34
```

Aufgabe P 8.3 *** (Einfach verkettete Liste, Hausaufgabe)

Betrachten Sie die beiden Vorlagen `VerketteListe.java` und `VerketteListeDemo.java`. Die Klasse `VerketteListeDemo` ist bereits fertig implementiert.

Vervollständigen Sie die Klasse `VerketteListe` so, dass folgende Beispielausgabe beim Ausführen von `VerketteListeDemo` (mit entsprechenden Nutzereingaben) gedruckt wird. In der Vorlage sind alle Stellen markiert, an denen Sie etwas einfügen müssen.

```
Neue int-Liste erstellt.
Was wollen Sie tun? (a)dd - (d)elete - (p)rint - (c)lose?
a
Welcher Wert soll hinzugefügt werden?
2
Listenausgabe: (2)
Was wollen Sie tun? (a)dd - (d)elete - (p)rint - (c)lose?
a
Welcher Wert soll hinzugefügt werden?
4
Listenausgabe: (4, 2)
Was wollen Sie tun? (a)dd - (d)elete - (p)rint - (c)lose?
p
Listenausgabe: (4, 2)
Was wollen Sie tun? (a)dd - (d)elete - (p)rint - (c)lose?
d
Listenausgabe: (4)
Was wollen Sie tun? (a)dd - (d)elete - (p)rint - (c)lose?
c
```

Aufgabe P8.4 ** (Software-Tests)

Implementieren Sie eine Klassenmethode `normDensity(double, double, double)`, welche die Parameter `x`, `my` und `sigma` besitzt und als Rückgabe einen double-Wert liefert. Die Methode soll folgende statistische Formel berechnen und das Ergebnis auf 4 Nachkommastellen runden:

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{(-\frac{1}{2}(\frac{x-\mu}{\sigma})^2)}, \quad \mu \in \mathbb{R}, \sigma \in \mathbb{R}_0^+$$

Verwenden Sie für die Implementierung die Bestandteile der Klasse *Math* (API). In dieser finden Sie auch eine Methode zum ganzzahligen Runden der Ergebnisse. Überlegen Sie sich, wie sie mithilfe der Methode auf vier Nachkommastellen genau runden können.

Da die Implementierung aufgrund der Komplexität der Formel fehleranfällig ist, testen Sie ihr Programm unter Verwendung von JUnit. Implementieren Sie hierfür folgende Testfälle und erweitern Sie ihre Methode bei Bedarf:

- $x = 0, \mu = 0, \sigma = 1$, erwarteter Rückgabewert: 0.3989
- $x = 4, \mu = 12, \sigma = 4$, erwarteter Rückgabewert: positiver Rückgabewert
- $\sigma = 0$, erwarteter Rückgabewert: `0.0`
- $\sigma < 0$, erwarteter Rückgabewert: `-1`

Aufgabe P8.5 *** (Test-Driven Development)

In dieser Aufgabe soll unter Anwendung von Test-Driven Development (TDD) ein Algorithmus implementiert werden, welcher Palindrome erkennt. Palindrome sind Wörter, die vorwärts wie rückwärts denselben Text ergeben, z.B. Hannah, Radar, Uhu.

Schreiben Sie hierzu eine Methode `istPalindrom(char[])`, welches ein `char`-Feld entgegen nimmt. Die Methode soll für dieses Feld prüfen, ob die einzelnen Zeichen zusammen ein Palindrom bilden oder nicht. Das Ergebnis soll entsprechend zurückgegeben werden.

Führen Sie den iterativen Entwicklungszyklus des TDD nacheinander mit folgenden Testfällen durch:

1.) Bei Übergabe einer `null`-Referenz soll `false` zurückgeliefert werden

2.) Für ein `char`-Feld mit nur einem Element liefert die Methode `true` zurück

3.) Die Methode erkennt längere Palindrome mit einer geraden Anzahl an Zeichen korrekt

4.) Die Methode erkennt längere Palindrome mit einer ungeraden Anzahl an Zeichen korrekt

5.) Es werden auch Wörter mit einer Mischung aus Groß- und Kleinbuchstaben korrekt erkannt

Hinweis 1: Verwenden Sie anonmye Arrays als Methodenargumente beim Testen, z.B. new char[ ]{'A', 'N', 'N', 'A'}.

Hinweis 2: Die Methode Character.toLowerCase(char) wandelt einen übergebenen Buchstaben in einen Kleinbuchstaben um.

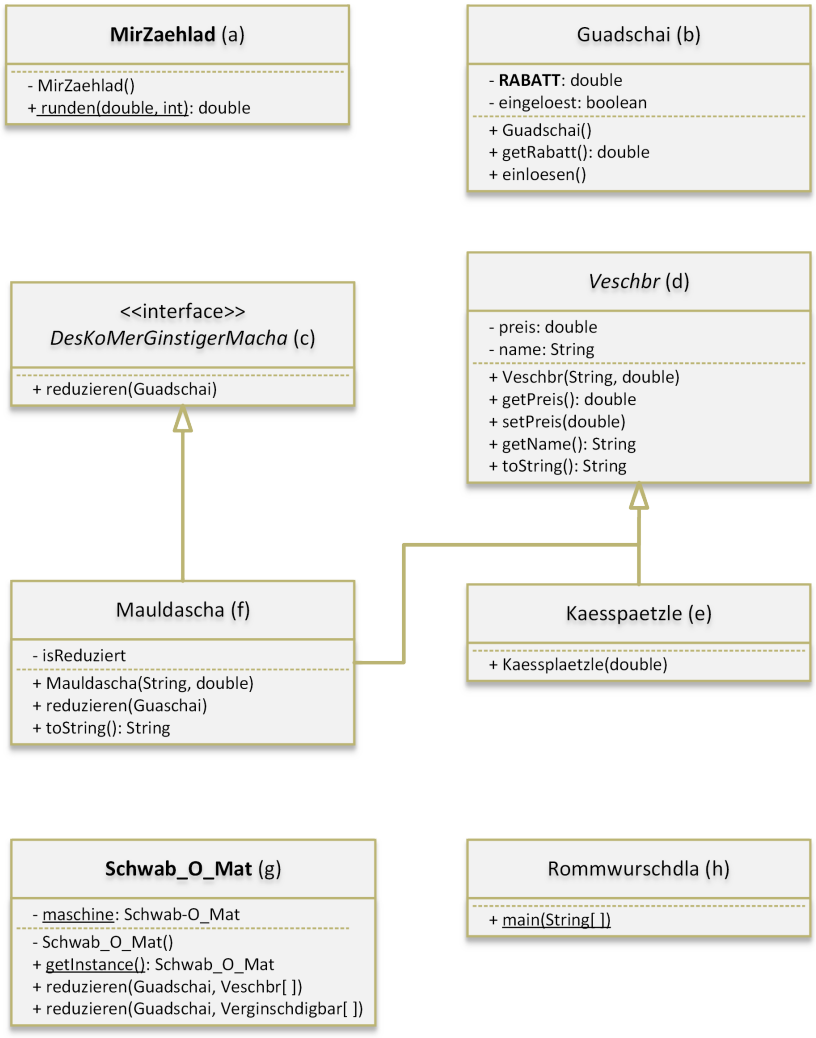
Aufgabe P 8.6 *** (Fortgeschrittene OOP)

Diese Aufgabe kombiniert diverse Konzepte aus der Objektorientierten Programmierung und soll Ihnen Anhaltspunkte geben, wie diese abseits der Veranstaltung sinnvoll in Kombination eingesetzt werden können. Insbesondere soll auch ein Ausblick auf die Möglichkeiten in Java gegeben werden, die im Rahmen der Veranstaltung nicht weitergehend behandelt wurden.

Prolog: *Mr schreibt's Johr 2019 n. Chr. 'S ganze Ländle isch von de Badner bsetzt... 'S ganze Ländle? Ha noe! Ganz Bade? Naai! Eis vo unbeugsam Schwobe bevölkerets Dörfle in Oschd-Badn hörd ned uuf, de Badner Widorstand z'leischte ond schbara weidr. Denn wer d'Zindhelzr nedd schbard wia d'Scheid, kommd nedd weid.*

Um den Geiz der *Volksfront von Schwaben (VvS)* zu brechen, schmieden die Kommandanten der Einheit *Badener ohne Grenzen (BoG)* einen ausgefuchsten Plan. Um die nächsten Züge des Widerstandes zu antizipieren, stellen die Feldherren ein sozioökonomisches Verhaltensmodell auf. Hierbei ist Ihre Unterstützung geboten!

Orientieren Sie sich bei der Bearbeitung der folgenden Aufgaben an den Beschreibungen sowie am gegebenen UML-Diagramm. **Im UML-Diagramm sind finale Elemente fett, abstrakte Elemente kursiv und statische Elemente unterstrichen markiert. Beachten Sie bei der Implementierung außerdem die Vererbungsstrukturen sowie die Sichtbarkeit von Methoden und Variablen.** Der Übersichtlichkeit halber ist es Ihnen gestattet, alternativ auch deutsche Bezeichnungen für die Klassen zu wählen (siehe Aufgabenstellung). **Es empfiehlt sich, zunächst die Bestandteile gemäß UML-Diagramm zu implementieren, ggfs. noch in Form von Platzhaltern und anschließend die Funktionalität hinzufügen.** Für das Nacharbeiten wird eine ausführliche Musterlösung bereitgestellt, die alle verwendeten Konzepte nochmals erläutert.



a) Um sich das Leben etwas einfacher zu gestalten und Rechenaufwand zu sparen, erstellen Sie eine neue Klasse *MirZaehlad* (dt. *wir zählen*), welche eine Klassenmethode bereitstellt, um Dezimalzahlen auf eine festgelegte Anzahl an Nachkommastellen zu runden. Stellen Sie sicher, dass von der Klasse keine Unterklassen oder Instanzen gebildet werden können, da diese keinen sinnvollen Nutzen hätten (*Hinweis: Wie muss der Konstruktor gestaltet werden, sodass keine Instanzen erzeugt werden können?*).

b) Ein essentieller Lebensinhalt der *VvS* stellt der Umgang mit Gutscheinen dar, die bei der Einlösung einen Rabatt auf ein Produkt gewähren. Modellieren Sie eine entsprechende Klasse *Guadschai* (dt. *Gutschein*). Dabei soll bei der Erzeugung eines Gutscheines dieser mit einem zufälligen Rabattwert im Bereich von 1% bis 10% generiert werden. Achten Sie darauf, dass nur ganze Prozentzahlen generiert werden. Zusätzlich besitzt jeder Instanz ein Flag `eingeloest`, welches anzeigt, ob ein Gutschein bereits verwendet worden ist. Ein eingelöster Gutschein liefert entsprechend einen Rabatt von 0%.

c) Nicht bei jedem Produkt lässt sich mithilfe eines Gutscheines etwas sparen. Um die reduzierbaren von den nicht-reduzierbaren Produkte zu unterscheiden, erstellen Sie ein Interface *DesKoMerGinstigerMacha* (dt. *reduzierbar*), mit welchem implementierende Klassen eine Methode zum Reduzieren des Preises erhalten.

d) Geld lässt sich bekanntlich nur sparen, wenn man es (zumindest teilweise) auch ausgibt. Modellieren sie hierzu eine Klasse *Veschbr* (dt. *Mahlzeit*) gemäß dem gegebenen URL-Diagramm, um das Konsumverhalten von Schwaben zu studieren. Beachten Sie, dass ein *Veschbr* zu unspezifisch ist, um davon eine Instanz zu erzeugen. Ergänzen Sie Getter, Setter sowie eine `toString`-Methode, welche den Namen und Preis des Produkts zurückgibt.

e) Bekanntlich nehmen Mitglieder der *VvS* als Hauptmahlzeit leckere *Kaessplaetzle* (dt. *Käsespätzle*) zu sich. Implementieren sie eine entsprechende Unterklasse von *Veschbr* und schreiben Sie einen passenden Konstruktor. Sie können davon ausgehen, dass alle Objekte den Namen *Käsespätzle* erhalten. Verwenden Sie zur Festlegung der Bezeichnung und des Preises den Superkonstruktor der Klasse.

f) Zur Kräftigung des Völkleins gibt es an jedem Sonntag außerdem *Mauldascha* (dt. *Maultaschen*) zu Essen. Und weil der Spaß am größten ist, wenn Arbeit und Hobby gemeinsam betrieben werden, lässt sich bei den Kosten für eine Einheit *Mauldascha* mithilfe von Gutscheinen auch noch etwas sparen. Klasse! Beachten Sie, dass der Preis für eine Einheit nur reduzierbar ist, wenn der Preis noch nicht zuvor reduziert und der Gutschein noch nicht verwendet worden ist. Ergänzen Sie ein entsprechendes Flag und implementieren Sie die geerbte Methode zum Reduzieren des Preises. Überschreiben Sie zusätzlich die geerbte `toString`-Methode derart, dass der String die zusätzliche Information enthält, ob das Produkt reduziert wurde oder nicht.

g) Findigen Pfennigfuchsern ist es gelungen, eine Maschine zu entwickeln, die es ermöglicht, einzelne Gutscheine auf mehrere Produkte anzuwenden: den *Schwab_O_Mat*. Diese Maschine ist einzigartig in der Welt, sodass es in Ihrem Programm lediglich eine Instanz des *Schwab_O_Mat* geben kann! Stellen Sie sicher, dass keine Unterklassen gebildet werden können und lediglich eine Instanz zur Laufzeit existiert, welche von jedem verwendet wird (Hinweis UML: `getInstance()`. (*Hinweis: Stellen Sie eine Instanz in Form der Klassenvariablen maschine zur Verfügung.*)

   Geben Sie bei der Erzeugung der Instanz *maschine* eine entsprechende Mitteilung auf der Konsole aus. Verwenden Sie hierzu einen statischen Initialisierer.

   Bei der Anwendung eines Gutscheins auf mehrere Produkte ist für jedes Produkt separat zu prüfen, ob dieses reduzierbar ist (Hinweis: Ein Produkt ist dann reduzierbar, wenn es das Interface *DesKoMerGinstigerMacha* implementiert hat). Nach Verwendung ist der Gutschein selbstverständlich zu entwerten. Da nicht immer von vornherein klar ist, ob ein Produkt reduzierbar ist, schreiben Sie für beide Fälle jeweils eine eigene Methode.

h) Nun geht es in die heiße Phase: Erstellen Sie eine Klasse *Rommwurschdla* (dt. *arbeiten*), in welcher Sie ihr Modell testen. Generieren Sie mehrere Gutscheine und Speisen und führen Sie entsprechende Rabattprozesse durch. Verwenden Sie in einem weiteren Schritt den *Schwab_O_Mat*, um Gutscheine mehrfach zu verwenden. Die Ergebnisse können Sie sich jeweils auf der Konsole ausgeben lassen.

Das Prog I-Team wünscht Ihnen viel Erfolg bei der Klausur!