Karlsruher Institut für Technologie

Programmieren – Wintersemester 2017/18

Architecture-driven Requirements Engineering (ARE)
`https://sdqweb.ipd.kit.edu/wiki/Programmieren`
Jun.-Prof. Dr.-Ing. Anne Koziolek

Übungsblatt 2

Ausgabe: 22.11.2017 – 13:00
Abgabe: 07.12.2017 – 06:00

- Achten Sie darauf, nicht zu lange Zeilen, Methoden und Dateien zu erstellen.
- Sie müssen bei Ihren Lösungen eine maximale Zeilenbreite von 120 Zeichen einhalten.
- Programmcode muss in englischer Sprache verfasst sein.
- Kommentieren Sie Ihren Code angemessen: So viel wie nötig, so wenig wie möglich.
- Die Kommentare können sowohl in englischer Sprache, als auch in deutscher Sprache geschrieben werden. Die Sprache innerhalb eines Übungsblattes muss jedoch einheitlich bleiben.
- Verwenden Sie keine Klassen der Java-Bibliotheken ausgenommen Klassen der Pakete `java.lang.*` und `java.io.*`, außer die Aufgabenstellung erlaubt ausdrücklich weitere Pakete.
- Achten Sie auf fehlerfrei kompilierenden Programmcode[1].
- Halten Sie alle Whitespace-Regeln ein.
- Halten Sie die Regeln zu Variablen-, Methoden- und Paketbenennung ein und wählen Sie aussagekräftige Namen.

Abgabemodalitäten

Die Praktomat-Abgabe wird am **Mittwoch, den 29. November 2017, um 13:00 Uhr**, freigeschaltet.

- Geben Sie Ihre Antworten zu Aufgabe A als `*.java`-Dateien ab.
- Geben Sie Ihre Antworten zu Aufgabe B als `*.java`-Dateien ab.
- Geben Sie Ihre Antworten zu Aufgabe C als `*.java`-Dateien ab.
- Geben Sie Ihre Antworten zu Aufgabe D als `*.java`-Dateien ab.

Achten Sie unbedingt darauf, diese Dateien bei der richtigen Aufgabe hochzuladen. Falsch hochgeladene Abgaben werden nicht bewertet.

Planen Sie für die Abgabe ausreichend Zeit ein, für den Fall, dass der Praktomat Ihre Abgabe wegen einer Regelverletzung ablehnt.

Einführung von Checkstyle
Beginnend mit diesem Übungsblatt überprüft der Praktomat Ihre Quelltexte während der Abgabe automatisiert auf die Einhaltung obenstehender Regeln. Falls eine Regel entsprechend markiert ist, wird der Praktomat die Abgabe zurückweisen, wenn die Regel verletzt ist. Andere Regelverletzungen führen zu Punktabzug. Sie können (und sollten) Ihre Quelltexte bereits während der Entwicklung auf die Regeleinhaltung überprüfen. Das Programmieren-Wiki (ILIAS) beschreibt, wie das funktioniert.

[1] Der Praktomat wird die Abgabe zurückweisen, falls diese Regel verletzt ist.

Öffentliche Tests
Bitte beachten Sie, dass das erfolgreiche Bestehen der öffentlichen Tests für eine erfolgreiche Abgabe dieses Blattes notwendig ist. Planen Sie entsprechend Zeit für Ihren ersten Abgabeversuch ein.

Objektorientierte Modellierung
Achten Sie darauf, dass Ihre Abgaben sowohl in Bezug auf objektorientierte Modellierung, als auch Funktionalität bewertet werden.

A Kontrollstrukturen (8 Punkte)

A.1 Schleifen in Java (6 Punkte)

In der Vorlesung haben Sie bereits verschiedene Schleifen kennengelernt, unter anderem auch die `for`-Schleife, die `while`-Schleife und die `do-while`-Schleife. In dieser Aufgabe geht es darum zu entscheiden, bei welchem Fall welche der drei Schleifen am geeignetsten ist. Implementieren Sie die drei Unteraufgaben in jeweils einer separaten Methode. Schreiben Sie dazu eine Klasse `Main`, die die `main`-Methode enthält. Die Ergebnisse der Unteraufgaben sollen jeweils in einer Zeile auf dem Terminal ausgegeben werden. Dazu können Sie die Methode `System.out.println()` nutzen. Die Anzahl der Nachkommastellen Ihrer Programmausgabe ist für diese Aufgabe nicht relevant.

A.1.1 Produkt

Schreiben Sie eine Methode, in welcher das Produkt aller positiven `Integer`-Zahlen im abgeschlossenen Intervall $[n_1, n_2]$ berechnet wird. Die Zahlen n_1 und n_2 sind auch zwei positive `Integer`-Zahlen, die Ihre Methode als Parameter entgegennimmt. Sie können davon ausgehen, dass sowohl n_1 und n_2 immer gültig sind, als auch stets $n_1 < n_2$ gilt. Geben Sie schließlich in der `main`-Methode beispielhaft das Ergebnis Ihrer Berechnung für $n_1 = 6$ und $n_2 = 11$ auf dem Terminal aus.

A.1.2 Countdown

Implementieren Sie in dieser Aufgabe einen Zähler, der ganze Zahlen von einer `Integer`-Zahl n_1 bis zur einer weiteren `Integer`-Zahl n_2 herunterzählt. Ihre Methode nimmt die Zahlen n_1 und n_2 als Parameter entgegen. Geben Sie bei jedem Zählschritt den aktuellen Wert des Zählers in einer einzigen Zeile separiert durch genau ein Leerzeichen auf dem Terminal aus. Sowohl die Zahl n_1, als auch die Zahl n_2 müssen in Ihrer Ausgabe vorhanden sein. Sie können davon ausgehen, dass sowohl n_1 und n_2 immer gültig sind, als auch stets $n_1 > n_2$ gilt. Zudem brauchen Sie keine Randfälle, wie z.B. einen Unterlauf, zu berücksichtigen. Geben Sie schließlich in der `main`-Methode beispielhaft das Ergebnis Ihrer Berechnung für $n_1 = 5$ und $n_2 = 0$ auf dem Terminal aus.

A.1.3 Zahlenfolge

Implementieren Sie eine Methode, in welcher beginnend mit einer positiven `Integer`-Zahl n_1 alle positiven ungeraden Zahlen, die kleiner als oder gleich einer weiteren positiven `Integer`-Zahl n_2 sind, in einer einzigen Zeile separiert durch genau ein Leerzeichen auf dem Terminal aus. Sind die Zahlen n_1 und n_2 auch positive ungerade Zahlen, werden sie ebenfalls ausgegeben. Ihre Methode nimmt die Zahlen n_1 und n_2 als Parameter entgegen. Sie können davon ausgehen, dass sowohl n_1 und n_2 immer gültig sind, als auch stets $n_1 < n_2$ gilt. Zudem brauchen Sie keine Randfälle, wie z.B. einen Überlauf, zu berücksichtigen. Geben Sie schließlich in der `main`-Methode beispielhaft das Ergebnis Ihrer Berechnung für $n_1 = 1$ und $n_2 = 14$ auf dem Terminal aus.

A.2 Objektorientierte Modellierung: Versicherung (2 Punkte)

In dieser Aufgabe soll vereinfacht eine Autoversicherung (Kfz-Haftpflichtversicherung) modelliert werden. Sie berechnet einen Versicherungsbeitrag VB mit VB = G + mW. In dieser Berechnungsvorschrift ist G ein Grundbetrag, der primär von der Anzahl der bisherigen Unfällen des Versicherungsnehmers abhängig ist (s. Tabelle 1). Der Faktor m wird mit dem konstanten Wert 0.5 belegt und W bezeichnet den Wert des Autos, der immer eine positive ganze Zahlen ist. Es sollen ausdrücklich nur die Fälle in Tabelle 1 berücksichtigt werden.

Unfälle	1	2	3	4
Grundbetrag G	33	64	85	112

Tabelle 1: Tabellarischer Überblick zu dem Grundbetrag in der Währung Euro

Schreiben Sie ein Programm, das den Versicherungsbeitrag für die unterschiedlichen Fälle berechnet. Geben Sie das Ergebnis für G = 85 (d.h. Anzahl der Unfälle ist 3) und W = 3000 auf dem Terminal aus. Dazu können Sie die Methode `System.out.println()` nutzen. Verwenden Sie für die Ausgabe die `main`-Methode der Aufgabe A.1.

B Konstruktoren (5 Punkte)

B.1 Konstruktordefinitionen (2 Punkte)

Die Klasse `Point` wird im zweidimensionalen kartesischen Koordinatensystem über die Koordinaten x und y von Typ `int` beschrieben. Diese Attribute sollen nur innerhalb der Klasse direkt angesprochen werden. Wählen Sie dafür einen geeigneten Modifikator. Eine Spezifikation der `main`-Methode ist nicht erforderlich. Ergänzen Sie diese Klasse um folgende Konstruktoren:

- 1. Konstruktor: Ein parameterloser Konstruktor soll alle Attribute der Klasse `Point` auf `0` setzen.
- 2. Konstruktor: Ein weiterer Konstruktor, der zwei Argumente vom Typ `int` mit dem gleichen Namen wie die der Attribute übergeben bekommt.
- 3. Konstruktor: Ein sogenannter Copy-Konstruktor (auch Kopierkonstruktor), der nur einen Parameter in Form einer Referenz vom Typ der eigenen Klasse besitzt. Ein erzeugtes Objekt hat die gleichen Werte wie das übergebene.

Anmerkung: In der Praxis kann es vorkommen, dass man ähnliche oder inhaltlich identische Objekte benötigt. Java bietet dazu zwei Alternativen an: (i) die Methode `clone()` der Klasse `Object`, die ein geklontes Objekt erzeugt und (ii) den Kopierkonstruktor (wie oben beschrieben), der eine Kopie eines Objektes erzeugt.

B.2 Objektorientierte Modellierung: Fortsetzung der Filmdatenbank (3 Punkte)

Die Modellierung einer fiktiven Filmdatenbank aus Übungsblatt 1 soll in dieser Aufgabe erweitert werden. Neben der Definition der Klassen *Series*, *Season* (deutsch: Staffel) und *Episode* und deren Attribute, sollen nun die Konstruktoren festgelegt werden. Auch bei dieser Aufgabe sind keine weiteren Methoden erfordert. Achten Sie darauf, dass Ihr Programm erfolgreich kompiliert. Verwenden Sie für diese Aufgabe die Musterlösung der Aufgabe D.1 des letzten Übungsblattes.

Der Konstruktor einer **Serie** nimmt einzig den *Namen* der Serie entgegen und weist diesen dem entsprechenden Attribut zu. Die restlichen Attribute bleiben undefiniert.

Der Konstruktor einer **Staffel** nimmt ein *Series*-Objekt entgegen sowie die *Nummer* der Staffel. Sowohl das *Serie*-Objekt, als auch die Staffelnummer werden dem entsprechenden Attribut zugewiesen. Die restlichen Attribute bleiben undefiniert.

Der Konstruktor einer **Episode** nimmt ein *Season*-Objekt entgegen sowie die *Nummer* der Episode. Sowohl das *Season*-Objekt, als auch die Episodennummer werden dem entsprechenden Attribut zugewiesen. Die restlichen Attribute bleiben undefiniert.

C Überladene Methoden (2 Punkte)

In Java ist es möglich Methoden zu überladen (method overloading). Dabei besitzt eine Klasse mehrere Methoden mit dem gleichen Namen, die sich in ihren Parametern unterscheiden. Analog dazu existieren auch überladene Konstruktoren (siehe Aufgabe B.1).
In dieser Aufgabe soll eine Klasse `Rectangle` definiert werden, die zwei Attribute a und b vom Typ `int` besitzt, die die Seitenlänge eines Rechtecks beschreiben. Im Konstruktor der Klasse werden die `int`-Werte zum Initialisieren übergeben. Es sollen zwei Methoden für die Berechnung des Flächeninhaltes eines Rechtecks implementiert werden. Zum einen eine parameterlose Objektmethode `surface`, zum anderen eine Klassenmethode (statische Methode), die die Objektmethode überlädt und eine Referenz vom Typ der eigenen Klasse übergeben bekommt. Eine Spezifikation der `main`-Methode ist nicht erforderlich.

D Objektorientierte Modellierung: Bankinstitut (5 Punkte)

In dieser Aufgabe soll vereinfacht ein Bankinstitut modelliert werden. Überlegen Sie sich bei dieser Aufgabe, ob es sich bei den jeweiligen Attributen um Klassen- oder Objektvariablen handelt. Erweitern Sie alle Klassen um geeignete `getter`- und/oder `setter`-Methoden. Sind `getter`- und/oder `setter`-Methoden für ein Attribut mit dem Namen `Attributname` sinnvoll vorhanden, nennen Sie die entsprechenden Methodennamen `getAttributname` und/oder `setAttributname`. Zum Beispiel lauten die `getter`- und `setter`-Methoden für ein Attribut `String name` `getName` und `setName`. **Wichtig**: Jede Entscheidung muss begründet werden. Schreiben sie dazu pro Methode ein Kommentar, der darlegt, warum Sie sich für oder gegen die `getter`-/`setter`-Methoden entschieden haben. Schreiben Sie schließlich zu den folgenden Klassen eine separate Klasse `Main` mit einer `main`-Methode. Diese Methode instanziiert mindestens ein beliebiges Konto (`Account`) mit einem Bankkunden als Kontoinhaber (`AccountHolder`). Geben Sie den Zustand des Kontos sowie des Kontoinhabers auf dem Terminal aus. Dazu können Sie die Methode `System.out.println()` nutzen.

D.1 Klasse `Account`

Implementieren Sie eine Klasse `Account`, die ein Konto darstellt. Die Klasse `Account` besitzt folgende Attribute: `int accountNumber` für die Darstellung der Kontonummer, `int bankCode` für die Darstellung der Bankleitzahl, die sich aus der IBAN ableiten lassen, und `int balance` für die Darstellung des Kontostandes. Die IBAN muss für die Aufgabenstellung nicht explizit berücksichtigt werden. Sie können annehmen, dass Kontonummer, Bankleitzahl und Kontostand positive ganze Zahlen sind.

Implementieren Sie einen geeigneten Konstruktor, der die Bankleitzahl entgegennimmt und diese initialisiert. Der Kontostand ist beim Initialisieren eines Kontos stets 0. Die Kontonummer wird dem Konstruktor nicht als Parameter übergeben, sondern richtet sich nach der Anzahl der bereits instanziierten Konten: Das erste Konto erhält die Kontonummer 0, das zweite Konto die Kontonummer 1 und das i-te Konto erhält die Kontonummer i-1. In dieser Aufgabe können Sie davon ausgehen, dass alle Eingaben gültig sind. Somit brauchen Sie keine Randfälle, wie z.B. einen Überlauf, zu berücksichtigen.

Implementieren Sie zudem folgende Methoden für die Klasse `Account`:

- `boolean withdraw(int amount)`: Diese Methode repräsentiert das Auszahlen. Dabei wird der Kontostand um den übergebenen Betrag (`amount`) verringert. Wird der Kontostand durch die Auszahlung negativ, findet die Auszahlung nicht statt, d.h. der Kontostand wird nicht geändert. In diesem Fall liefert die Methode den Rückgabewert `false`, andernfalls `true`.
- `void deposit(int amount)`: Diese Methode repräsentiert das Einzahlen. Dabei wird der Kontostand um den übergebenen Betrag (`amount`) erhöht.
- `void transfer(Account account, int amount)`: Diese Methode repräsentiert das Überweisen. Dabei entspricht `account` dem Zielkonto und `amount` dem Betrag, der überwiesen werden soll. Der Kontostand des (Quell-)Kontos, von dem der Betrag überwiesen wird, wird um den übergebenen Betrag verringert und der Kontostand des Zielkontos, worauf der Betrag überwiesen wird, wird um diesen Betrag erhöht. Sie können hier davon ausgehen, dass nur gültige Überweisungen durchgeführt werden (z.B. dass die

Kontonummer des Zielkontos immer gültig ist oder dass der Kontostand des Quellkontos durch diese Überweisung nicht negativ wird).

D.2 Klasse `AccountHolder`

Implementieren Sie eine Klasse `AccountHolder` zur Darstellung der Kontoinhaber. Jeder Kontoinhaber besitzt nur ein einziges Konto - `Account account`. Jeder Kontoinhaber besitzt zudem folgende Attribute: `String firstName` und `String lastName` und `int accountHolderNumber`. Implementieren Sie einen geeigneten Konstruktor, der alle Attribute genau in dieser **Reihenfolge**, d.h. `Account account`, `String firstName`, `String lastName`, `int accountHolderNumber` entgegennimmt und diese initialisiert.