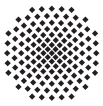


Universität Stuttgart

Übungsblatt 7

Programmierung und Softwareentwicklung (WS 17/18)

Abgabe: Freitag, 22.12.2017, 23:55 Uhr — Besprechung: ab Montag, 8.1.2018

Bitte lösen Sie die Übungsaufgaben in **Gruppen von 3 Studenten** und wählen EINEN Studenten aus, welcher die Lösung in ILIAS als **Gruppenabgabe** (unter Angabe aller Gruppenmitglieder) einstellt. Bitte erstellen Sie dazu einen **Header**, welcher die Namen der Studenten, die Matrikelnummern und die E-Mail-Adressen enthält.

Die Aufgaben, bei denen Quellcode abzugeben ist, sind mit Impl gekennzeichnet. Bitte beachten Sie die Hinweise zu den Implementierungsaufgaben, die in ILIAS verfügbar sind.[1] Achten Sie besonders darauf, dass Sie zu jeder Klasse und Methode JavaDoc-Kommentare erstellen.

Dieses Übungsblatt beinhaltet 4 Aufgaben mit einer Gesamtzahl von 40 Punkten.

Aufgabe 1 Impl Testen mit JUnit *[Punkte: 10]*

Laden Sie sich das im ILIAS hochgeladene Java-Projekt herunter und bearbeiten Sie die folgenden Aufgaben. Beachten Sie dabei, dass Sie dieses Projekt für mehrere Aufgaben auf diesem Blatt benötigen.

(a) *(4 Punkte)* Vervollständigen Sie die gegebene Klasse `BankAccount` gemäß der folgenden Beschreibung:

- Das Attribut `balance` vom Typ `double` speichert den Kontostand in Euro.
- Ein neues Konto kann nur unter Angabe eines initialen Kontostands eröffnet werden. Falls der angegebene initiale Kontostand negativ ist, dann wird ein neues Konto mit initialem Kontostand 0.0 eröffnet.
- Die Methode `deposit` führt eine Einzahlung durch, indem sie den angegebenen Betrag auf das Konto verbucht. Wenn dieser Methode eine negative Zahl übergeben wird, dann verändert die Methode den Kontostand nicht. Die Methode gibt keine Rückgabe.
- Die Methode `withdraw` überprüft zunächst, ob auf dem Konto genug Geld vorhanden ist, um den angegebenen Betrag abheben zu können. Ist dies der Fall, dann wird der Betrag vom aktuellen Kontostand abgezogen und die Methode gibt `true` zurück. Ansonsten soll der Kontostand unverändert bleiben und `false` zurückgegeben werden. Wird ein negativer Wert übergeben, so bleibt der Kontostand ebenfalls unverändert und es wird `false` zurückgegeben.

(b) *(6 Punkte)* Vervollständigen Sie die gegebene Klasse `TestBankAccount` zu einer `JUnit`-Testklasse für die `BankAccount` Klasse. Legen Sie darin mindestens einen Testfall je Methode der Klasse `BankAccount` an und testen Sie damit Ihre Implementierung aus Teilaufgabe a). Dabei soll jede Verzweigung innerhalb der einzelnen Methoden getestet werden.

Aufgabe 2 Impl Zahlensysteme II *[Punkte: 10]*

Laden Sie sich das im ILIAS hochgeladene Java-Projekt herunter und bearbeiten Sie die folgenden Aufgaben. Beachten Sie dabei, dass Sie dieses Projekt für mehrere Aufgaben auf diesem Blatt benötigen.

Vervollständigen Sie die Klasse `NumeralSystems` so, dass sie Methoden zur Verfügung stellt, mit denen man positive Zahlen zwischen verschiedenen Zahlensystemen umwandeln kann. Dabei sollen auf jeden Fall das Binär-, das Dezimal- und das Hexadezimalsystem (also die Basen 2, 10 und 16) unterstützt werden. Weitere Zahlensysteme sind jedoch willkommen.

[1] `https://ilias3.uni-stuttgart.de/goto_Uni_Stuttgart_fold_1318001.html`

Erweitern Sie `NumeralSystems` zusätzlich um ein Hauptprogramm, das vom Nutzer zunächst die Basis des Ausgangszahlensystems einliest und überprüft, ob dieses Zahlensystem unterstützt wird. Ist das Zahlensystem gültig, so soll über die Konsole eine Zahl im gewählten Zahlensystem eingegeben werden. Wurde die Zahl korrekt eingegeben, so soll anschließend das Zielzahlensystem gewählt werden und danach die gewählte Repräsentationsart ausgegeben werden. Anschließend kann der Nutzer eine weitere Zahl umwandeln lassen. Das Programm soll (auch bei einer Fehleingabe des Nutzers) so lange laufen, bis der Nutzer es explizit, z.B. durch die Eingabe einer „-1“ bei der Systemauswahl, beendet. Verwenden Sie hierfür eine geeignete Menüstruktur und behandeln Sie alle möglichen Ausnahmen, die durch falsche Nutzereingaben geworfen werden können.
Beachten Sie, dass Sie die Methoden `parseInt(String s, int radix)`, `toString(int i, int radix)`, sowie `toBinaryString(int i)`, `toOctalString(int i)` und `toHexString(long i)` aus den Wrapper-Klassen `Byte`, `Short`, `Integer` und `Long` nicht verwenden dürfen.

Aufgabe 3 Impl Generics, Sortieren und Vergleichen *[Punkte: 10]*

Laden Sie sich das im ILIAS hochgeladene Java-Projekt herunter und bearbeiten Sie die folgenden Aufgaben. Beachten Sie dabei, dass Sie dieses Projekt für mehrere Aufgaben auf diesem Blatt benötigen.

(a) *(5 Punkte)* Implementieren Sie die Methode `sort` der Klasse `JavaGenerics`. Die Methode bekommt eine Liste mit Elementen eines generischen Datentyps `T` übergeben und gibt eine Liste mit Elementen des Datentyps `T` zurück. Über den Datentyp `T` ist nur bekannt, dass dieser die Schnittstelle `Comparable` implementiert. Das bedeutet, dass zwei Elemente des Datentyps `T` miteinander verglichen werden können.
Die Methode `sort` soll nacheinander die Elemente aus der übergebenen Liste betrachten und nacheinander jedes dieser Elemente an der „richtigen“ Stelle in der Ergebnisliste einfügen, sodass diese am Ende alle Elemente der ursprünglichen Liste, in aufsteigender Reihenfolge sortiert, enthält. Beachten Sie, dass jedes Element der übergebenen Liste nur einmal betrachtet werden darf und die übergebene Liste, sowie deren enthaltene Elemente nicht verändert werden dürfen!
Hinweis: es dürfen keine von Java bereitgestellten Sortierfunktionen verwendet werden.

(b) *(5 Punkte)* Gegeben sei folgendes Klassendiagramm der Klasse `Song`:

Song
-singer: String -title: String -year: int
+Song(String, String, int) +getSinger(): String +getTitle(): String +getYear(): int

Ergänzen Sie die Klasse `Song` so, dass sie das gegebene Klassendiagramm implementiert. Erweitern Sie anschließend die Klasse `Song` so, dass sie die Schnittstelle `Comparable` implementiert. Für den Vergleich von zwei Elementen `song1` und `song2` soll folgende Ordnungsrelation gelten: `song1` $<$ `song2` gdw.[2]

- `song1.singer` $<$ `song2.singer`
- `song1.singer` $=$ `song2.singer` $\wedge$ `song1.title` $<$ `song2.title`
- `song1.singer` $=$ `song2.singer` $\wedge$ `song1.title` $=$ `song2.title` $\wedge$ `song1.year` $>$ `song2.year`

`song1` $=$ `song2` gdw.

- `song1.singer` $=$ `song2.singer` $\wedge$ `song1.title` $=$ `song2.title` $\wedge$ `song1.year` $=$ `song2.year`

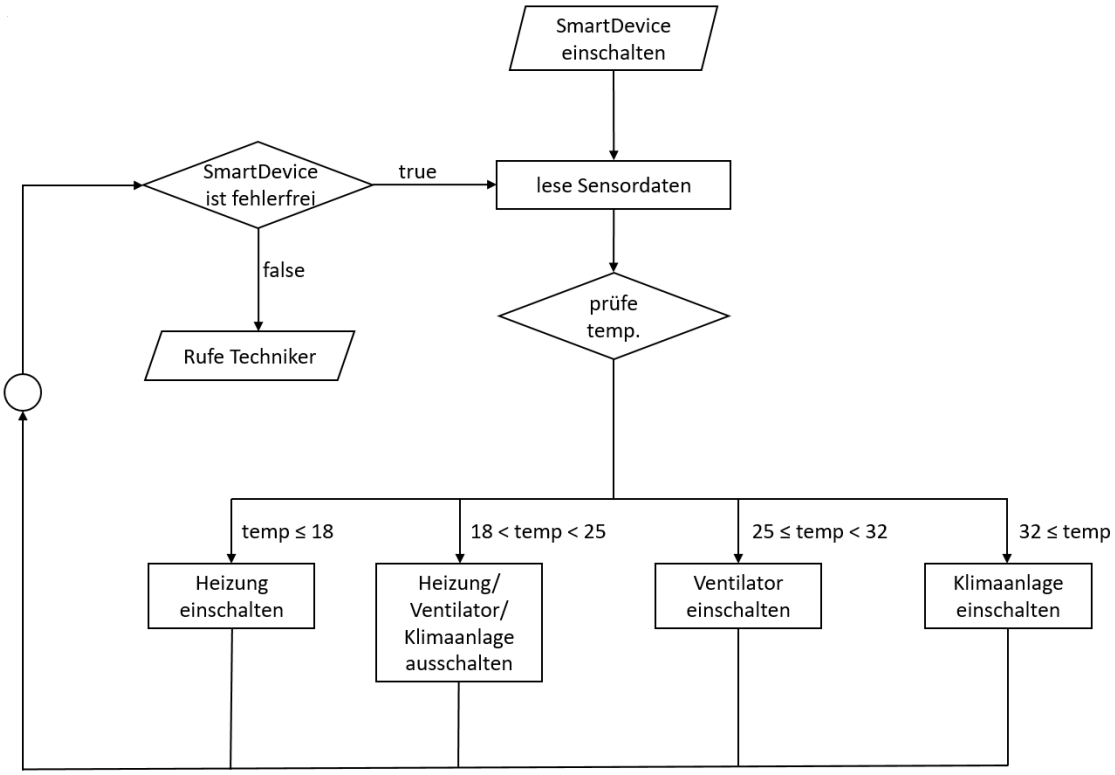
`song1` $>$ `song2` sonst.

Hierbei sind $<$ bzw. $>$ in Bezug auf zwei Zeichenfolgen als lexikalische Ordnungsrelation zu verstehen. Es gilt also zum Beispiel: Alpha $<$ Bravo.

[2] Abkürzung für: genau dann, wenn

Aufgabe 4 Flussdiagramm *[Punkte: 10]*

Formulieren Sie folgendes Flussdiagramm in Pseudo-Code. Verwenden Sie dabei die in der Vorlesung eingeführte Pseudo-Code Variante.



Zusätzlich zu den in der Vorlesung vorgestellten Konstrukten IF-THEN und IF-THEN-ELSE können Sie auch, wie im Folgenden exemplarisch dargestellt, ELSE-IF Konstrukte verwenden.

```
IF condition THEN
  statement-1
ELSE IF condition THEN
  statement-2
ELSE
  statement-3
ENDIF
```