Lehrstuhl für Angewandte Informatik I
Prof. Dr. Bernhard Westfechtel
Sandra Greiner

WS 2017/18
30. November 2017

Konzepte der Programmierung

Übung 7 – Abgabe: 14.12.2017

Aufgabe 7.1 (Sichtbarkeit und Lebensdauer)

Betrachten Sie folgendes Java-Programm:

```
1  public class Sichtbarkeiten
2  {
3      private int x = fak(5);
4      private static String name = "Blub";
5
6      public static void setzeName(int x)
7      {
8          if (x < 5)
9          {
10             name = "Fisch";
11         }
12         else
13         {
14             String name = "Philipp";
15         }
16     }
17
18     private int fak(double name) {
19         int sum = 1;
20         if (name > 1)
21         {
22             this.name = "why";
23             sum = (int) name * fak(--name);
24         }
25         x = 3;
26         for (int x = 0; x < 10; x++)
27         {
28             int a = 20;
29         }
30         return sum;
31     }
32 }
```

(a) Geben Sie durch Angabe entsprechender Zeilen den **Sichtbarkeitsbereich und die Lebensdauer** aller Variablen und Parameter an.

(b) Für welche Variable weichen **Lebensdauer** und **Sichtbarkeitsbereich** voneinander ab und warum?

(c) **(Bonus)** Welche Besonderheit gilt für die Methode `fak()`?

(d) **(Bonus)** Wie ändern sich Sichtbarkeit und Lebensdauer der einzelnen Variablen, wenn Objektmethoden statt Klassenmethoden verwendet werden.

(4+1 = 5 Punkte)

Aufgabe 7.2 (Kaffeevollautomat – Methodenaufrufe)
Diese Aufgabe übt nochmals den Umgang mit Objekten und Methoden ein. Insbesondere der Aufruf von (Objekt-)methoden wird hier nochmals verdeutlicht. Außerdem werden Aufzählungstypen und die `switch`-Anweisung wiederholt.

Wir implementieren in dieser Aufgabe einen `Kaffeeautomat`, die die Funktionsweise eines Kaffeevollautomaten simuliert. Der Kaffeevollautomat soll, bevor er Kaffee zubereitet, prüfen, ob alle benötigten Ressourcen, also Wasser und Bohnen zur Verfügung stehen. Außerdem darf der Satzbehälter noch nicht voll sein. Alle Ressourcen (auch der Satzbehälter) sind *ganzzahlig* und werden portionsweise, also pro Tasse, verwaltet. Um eine Tasse Kaffee zu bekommen, werden die Wassermenge und die Bohnenmenge entsprechend der Kaffeesorte verringert und die Kaffeesatz-Portionen um die Bohnenmenge erhöht.

Beim Ein- und Ausschalten wird der Automat jeweils in einem Spülvorgang gereinigt und die Wassermenge reduziert. Die Reinigung kann auch manuell vom Benutzer nach dem Brühen eines Kaffees durchgeführt werden.

Achten Sie bei der Lösung auf eine angemessene JavaDoc Dokumentation der Methoden. (d.h. die Signatur wird explizit mit `@param` `@return` angegeben)

(a) Erstellen Sie zunächst einen Aufzählungstyp `Kaffeesorte`, der die Werte `ESPRESSO`, `KAFFEE_NORMAL` und `KAFFEE_LANG` enthält.

(b) Legen Sie in der Klasse `Kaffeeautomat` 3 initialisierte Objektfelder an, die die Anzahl der Bohnen, die Wassermenge und den Inhalt des Satzbehälters repräsentieren. Die Bohnen und das Wasser sollen zunächst voll sein, der Satzbehälter leer.
Nehmen Sie an, dass der Zahlenwert 0 jeweils leer und der Zahlenwert 10 jeweils einen vollen Behälter repräsentiert.
Außerdem soll jede Instanz des Automaten zunächst ausgeschaltet sein.

(c) Schreiben Sie eine Methode `reinige()`, die die Wassermenge um eine Einheit reduziert und auf der Konsole die Meldung ausgibt `''Reinigung im Gang...''`.

(d) Schreiben Sie zwei Methoden zum Einschalten (`schalteEin()`) und Ausschalten (`schalteAus()`) des Automaten. Beide Methoden sollen am Ende die Methode `reinige()` aufrufen und somit den Reinigungsprozess simulieren.

(e) Implementieren Sie drei boolsche Methoden, die prüfen, ob genügend Wasser (1. Methode) bzw. Bohnen (2. Methode) vorhanden sind und ob noch genug Platz im Satzbehälter (3. Methode) ist für die gewünschte Kaffeesorte. Den Methoden wird die gewünschte Kaffeesorte übergeben.
Geben Sie zusätzlich eine Fehlermeldung aus, sobald von einer Ressource nicht genug vorhanden ist.
Die einzelnen Kaffeesorten benötigen folgende Ressourcen:

- Espresso: 3x Bohnen, 1x Wasser
- Kaffee normal: 1x Bohnen, 1x Wasser
- Kaffee lang: 1x Bohnen, 2x Wasser

Der Satzbehälter wird wie oben beschrieben um die benötigte Menge an Bohnen erhöht.

(f) Implementieren Sie eine weitere boolsche Methode `macheKaffee()`. Falls der Automat an ist, prüft diese Methode **unter Benutzung der obigen Methoden**, ob alle Ressourcen vorhanden sind. Die gewünschte Kaffeesorte wird ihr übergeben. Dann bereitet sie EINE Tasse dieser Sorte zu, indem sie einen String mit der jeweiligen Kaffeesorte (z. B. „Espresso

zubereitet.") ausgibt und `true` zurückgibt. Die Ressourcen werden entsprechend reduziert und der Satzbehälter entsprechend gefüllt. Sind nicht alle Ressourcen vorhanden, soll die Methode `false` zurückgeben.
Benutzen Sie für die Zubereitung eine switch-Anweisung.

(g) Schreiben Sie eine main-Methode, in der zuerst ein Kaffeeautomat (eine Instanz der Klasse `Kaffeeautomat`) erzeugt wird. Schalten Sie den Automaten ein. Dann wird die gewünschte Anzahl an Tassen Kaffee und deren Sorten vom Benutzer erfragt. Diese sollen einzeln nacheinander zubereitet werden mit der Methode `macheKaffee()`. Steht eine Ressource nicht mehr zur Verfügung, soll die Methode abbrechen.

(h) **[optional]** Mögliche Erweiterung der Klasse

- Schreiben Sie einen Konstruktor, dem Initialwerte für die Anzahl der Bohnen und für die Wassermenge übergeben werden. Der Satzbehälter ist anfangs aber immer leer.
- Lassen Sie Behälter **stufenweise** wieder füllen, indem Sie geeignete Methoden schreiben. Ihnen soll die Anzahl übergeben werden, um wie viel der jeweilige Behälter gefüllt wird.
- Eine `toString()-Methode`, die den aktuellen Zustand (Füllmengen) auf der Kommandozeile ausgibt.
- Fügen Sie einen Milchbehälter zum Kochen von Cappuccino oder Latte Macchiato hinzu.

(1+2+1+1+3+2+2 = 12 Punkte)

Aufgabe 7.3 (Seiteneffekte)
Machen Sie sich mit dem Programm `ProzeduralesProgrammMitSeiteneffekten.java` vertraut (zu finden bei dem aktuellen Übungsblatt im eLearning). Es berechnet im **prozeduralen** Programmierstil durch mehrere, voneinander ausgelöste Prozeduraufrufe einen Ergebnisstring, der in der globalen Variablen `s4` gespeichert wird und eine Ganzzahl (int) für die globalen Variablen `i1` und `i2`. Das Speichern und Auslesen der Zwischenergebnisse wird durch **Seiteneffekte** realisiert.

(a) Schreiben Sie eine neue Klasse `ProgrammOhneSeiteneffekte`, die einem **funktionalen** Programmierstil folgt und sich folgendermaßen von der prozeduralen Variante unterscheiden soll:

- Auf **alle** globale Variablen soll verzichtet werden.
- Anstatt globaler Variablen sollen in den Methodenrümpfen lokale Variablen verwendet werden.
- Ersetzen Sie *Prozedur*aufrufe durch *Funktions*aufrufe: Zwischenergebnisse sollen der jeweils aufrufenden Methode mit Hilfe von Rückgabewerten mitgeteilt werden. Passen Sie die Signaturen der Methoden entsprechend an und fügen Sie eine passende `return`-Anweisung hinzu. Verarbeiten Sie die Zwischenergebnisse in der aufrufenden Methode weiter.

Der einzige verbleibende Seiteneffekt soll die Kommandozeilen-Ausgabe (`Out.println`) im Hauptprogramm sein.

(b) Klassifizieren Sie alle Objektmethoden aus Aufgabe 7.2 (des Kaffeeautomaten) hinsichtlich dieser Eigenschaft. Welche Methoden sind frei von Seiteneffekten und welche nicht? Falls nicht, an welcher Stelle tritt der Seiteneffekt auf?

(2+2 = 4 Punkte)

Aufgabe 7.4
Erweitern Sie die Klasse `DoubleArrays` von Blatt 3 um eine weitere Klassenmethode: Benutzen Sie in dieser Aufgabe KEINE `while`-Schleifen mehr.

Die Methode `istEnthalten()` bekommt ein 2D Array und eine (Komma-)Zahl übergeben. Das Array soll durchlaufen werden und der Wert `true` zurückgegeben werden, wenn die Zahl enthalten ist. Schreiben Sie ihre Methode so, dass **unterschiedlich lange** Zeilen berücksichtigt werden.

(**2 Punkte**)

Aufgabe 7.5 (Veranschaulichung der Rekursion)
In der Vorlesung wurde bereits **Rekursion** erklärt und angewendet.

Mit der folgenden Aufgabe wollen wir in die Prinzipien der Rekursion einsteigen. Als Hilfestellung zur Einführung können Sie die folgenden (auch im E-Learning) verlinkten Internetseiten benutzen.

- Bildliche Darstellung: 'Was ist Rekursion'[1]
- Video zur Ausführung der Fakultätsfunktion[2]
- Erklärung der rekursiven Fakultätsberechnung für Schüler[3]

Die erste Aufgabe zur Verdeutlichung der Rekursion lautet wie folgt:

In New York City verlaufen die Straßen über große Gebiete hinweg rechtwinklig und nahezu äquidistant zueinander. Die Kreuzungen seien mit zweidimensionalen Koordinaten bezeichnet. Der Kreuzungspunkt $(0; 0)$ bezeichnet die Stadtmitte.

Eine gängige Frage die man sich in New York City stellen kann ist:

Wieviele kürzeste Wege gibt es von einer gegebenen Kreuzung $(x; y)$ *zur Stadtmitte?*

Die folgende rekursive Methode bietet einen Antwort auf die Frage:

```
1  public class NYC {
2
3      public static int berechneWegezahl(int x, int y) {
4          if (x == 0 || y == 0) return 1;
5          else {
6              int wegeLaengs = 0;
7              if (y < 0) wegeLaengs = berechneWegezahl(x, y + 1);
8              else wegeLaengs = berechneWegezahl(x, y - 1);
9
10             int wegeQuer = 0;
11             if (x < 0) wegeQuer = berechneWegezahl(x + 1, y);
12             else wegeQuer = berechneWegezahl(x - 1, y);
13             return wegeLaengs + wegeQuer;
14         }
15 }   }
```

Hinweis: Zur Lösung der Aufgabe a und b empfiehlt sich der Einsatz des Debuggers.

[1] `http://www.pohlig.de/Unterricht/Inf2003/Tag15/13.1_Was_ist_eine_Rekursion.htm`
[2] `https://www.youtube.com/watch?v=PORo1ut9kMs`
[3] `http://www.gym1.at/schulinformatik/aus-fortbildung/fachdidaktik/vk-01/jahresplan-pohlig/Tag29/Fakultaet.htm`

(a) Geben Sie eine Folge von **Aufrufstapeln** (s. Kapitel Rekursion) für die Abarbeitung des Aufrufs `berechneWegezahl(2,-1)` an.

(b) Geben Sie ein **Ausführungsprotokoll** (s. Kapitel Rekursion) für die Abarbeitung des Aufrufs `berechneWegezahl(3,-2)` an.

(c) Um welche Art von Rekursion handelt sich bei dieser Methode? Klassifizieren Sie die Methode hinsichtlich der Kriterien *linear*, *direkt*, *schlicht* (Dies schließt eine kurze Begründung mit ein).

(3+2+1 = 6 Punkte)