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

WS 2017/18
14. Dezember 2017

Konzepte der Programmierung

Blatt 9

Abgabe: 11.01.2017

Aufgabe 9.1 (Methodenaufrufe und Überladen)

Bearbeiten Sie diese Aufgabe bitte handschriftlich.

Gegeben sei die folgende Java-Klasse mit vier Methoden, deren Implementierung für diese Aufgabe uninteressant ist.

```
1 public class Methodenaufrufe {
2     public static long rechne(long x, int y, int z) { ... } // Methode 1
3     public static long rechne(int x, int y) { ... }         // Methode 2
4     public static String rechne(int x, String y) { ... }    // Methode 3
5     public static int schreibe(int x, String y) { ... }     // Methode 4
6 }
```

Geben Sie an, ob die folgenden Methodenaufrufe erlaubt sind. Bei erlaubten Methodenaufrufen geben Sie die Nummern der Methoden (1 bis 4) an, die für die Berechnung verwendet werden. Bei nicht erlaubten Methodenaufrufen geben Sie den Grund an.

(a) `long a = rechne('a', 'b', 'c');`

(b) `boolean b = rechne(1l, 2, 3) == schreibe(11, "neun");`

(c) `long c = rechne(1, 2, 3.0);`

(d) `long d = rechne(1, 2, rechne(3, 4));`

(e) `long e = rechne(8, "drei");`

(f) `int f = schreibe(schreibe(schreibe(1, "a"), "t"), "s");`

(**3 Punkte**)

Aufgabe 9.2 (Wiederholung – Arrays und Rekursion)

Implementieren Sie in der Klasse `DoubleArrays2D` eine **iterative** und eine **rekursive** Methode, die jeweils den Inhalt eines übergebenen 2D-Array transponieren und das transponierte Array zurückgeben.

In der rekursiven Methode sollen keine Schleifen verwendet werden. Das Originalarray, das transponiert werden soll, soll jeweils unverändert bleiben.

(**4 Punkte**)

Aufgabe 9.3 (Plätzchen – Aufzählungstypen, Arrays und Klassen mit Struktur)

Wir erweitern unsere Plätzchen-Aufgabe um die nachfolgenden Bestandteile. Überprüfen Sie anschließend Ihre Implementierun mit der `Main`-Klasse, die im e-Learning bereitgestellt wird.

(a) Die `PlaetzchenDose` soll die folgenden weiteren Methoden erhalten:

- `verringere(...)`, der eine Anzahl an Plätzchen übergeben wird, die aus der Dose entnommen wird. Falls mehr entnommen werden soll als vorhanden, muss kein Fehler ausgegeben werden. Der Bestand ist dann lediglich 0.
- `toString(...)`. Diese Methode wird geerbt von der Klasse `Object` und ist für jede beliebige Klasse vorhanden. Sie gibt **immer** einen String zurück und bekommt nichts übergeben.

  Wir überschreiben (**Redefinieren**) sie, indem wir darin unser gewünschtes Verhalten implementieren. Sie soll einen formatierten String zurückgegeben, der die Eigenschaften der Klasse nennt, also *welche* Sorte von Plätzchen darin enthalten ist, wie viele sich gerade darin befinden und insgesamt Platz haben.

(b) Implementieren Sie eine Klasse `Mensch`. Jeder Mensch kann eine Reihe von Plätzchendosen besitzen. Diese merkt er sich in einem Array mit Platz für fünf verschiedene Dosen.
Ein Mensch verändert den Bestand der Plätzchen, wenn er Plätzchen backt und nascht. Dafür und zum Füllen des Plätzchenarrays benötigen Sie die folgenden Methoden:

- `fuegeEin(...)`, die eine Plätzchendose übergeben bekommt. Diese wird an der nächsten freien Stelle im Array eingefügt. Sollte das Array schon voll sein, wird ein doppelt so großes Array erzeugt und die schon vorhandenen Dosen darin gespeichert.
- `backe(...)`, die den Plätzchentyp und die Anzahl, wie viel gebacken wurde, übergeben bekommt. Sie sucht im bestehenden Plätzchenvorrat eine Dose, die den gleichen Typ beinhaltet und füllt sie. Wenn keine passende Dose vorhanden ist oder die Dose dabei voll wird, wird eine neue Plätzchendose angelegt (und mit dem Rest befüllt). Der maximale Inhalt der neuen Dose soll dabei so gewählt werden, dass noch Platz für weitere Plätzchen ist.
- `nasche(...)`, die den Plätzchentyp und die Anzahl übergeben bekommt. Suchen Sie im Array nach einer passenden Dose. Falls danach die Dose leer ist, wird sie aus dem Array entfernt. Achten Sie darauf, dass Sie eventuelle Lücken schließen, wenn Sie den Vorrat in der Mitte des Arrays löschen.
- `toString()`, die den gesamten Plätzchenvorrat des Menschen als formatierten String zurückgibt unter Nutzung der `toString(...)`-Methode der `PlaetzchenDose`.

(3+13 = 16 Punkte)

Aufgabe 9.4
Statt wie in der vorhergehenden Aufgabe die Keksdosen des Menschen in einem Array zu verwalten, bietet sich dafür eine *dynamische Datenstruktur* an. Dies erspart den Umgang mit der statischen Größe eines Arrays.

Man könnte die Dosen stattdessen stapeln. Die erste Dose befindet sich unten und weitere Dosen werden danach darauf gestellt. Man kann allerdings nur die oberste Dose erreichen und aus dieser naschen.

Um dieses Verhalten zu realisieren, erstellen Sie eine Klasse `DosenStapel`, die Dosen speichern kann.

Gegeben Sei folgende Klasse:

```
public class DosenStapel {
    private Element oben;
    class Element {
        private PlaetzchenDose dose;
        private Element unteres;

        public Element(PlaetzchenDose d, Element u) {
            dose = d; unteres = u;
        }
    }
}
```

(a) Implementieren Sie nun folgende Bestandteile in dieser Klasse, die sich gemäß dem Prinzip eines Stapels verhalten.

- einen **Konstruktor**, dem eine Dose übergeben wird und als oberstes Element eingefügt wird und einen expliziten leeren Konstruktor, sowie die Methoden:
- `fuegeEin(...)`, der eine Dose übergeben wird und an geeigneter Stelle platziert wird.
- istLeer(...), die angibt, ob sich keine Elemente (mehr) auf dem Stapel befinden.
- `gibOben(...)`, die die oberste Dose zurückgibt, aber auf dem Stapel lässt.
- `entferne(...)`, die die oberste Dose zurückgibt (unter Nutzung von `gibOben()`) und vom Stapel nimmt. Falls der Stapel leer ist, wird zusätzlich ein Fehler auf der Konsole ausgegeben.

(b) Welches andere Prinzip kann man zur Verwaltung von Datenstrukturen benutzen, das genau gegenteilig funktioniert? Wie nennt man die entsprechende Datenstruktur und was müsste man an der Implementierung des Stapels ändern, um das andere Prinzip zu ermöglichen? *Sie können die Antwort dazu im Quelltext mit angeben.*

(4+2 = 6 Punkte)

Aufgabe 9.5 (Memory)
In dieser Aufgabe soll eine Basis-Implementierung des Spiels „Memory“ implementiert werden. Sie wird in den Übungssitzungen Schritt für Schritt erweitert bis das Spiel vollständig realisiert ist. Für die Realisierung kommen nach und nach wichtige objektorientierte Konzepte zum Einsatz.

Die Karten des Spiels beinhalten `char`-Zeichen. Jede Karte gibt es entsprechend auf dem „Spielfeld” genau zweimal. Implementieren Sie die folgenden Komponenten in einem neuen BlueJ-Projekt:

(a) Eine Klasse `Karte`, bestehend aus privaten Objektfeldern für ein Symbol (Datentyp `char`) und einem Wahrheitswert, der angibt, ob die Karte aufgedeckt ist.

Der Konstruktor für `Karte` ist parameterlos. Die Karte soll zunächst nicht aufgedeckt sein. Das Symbol von instantiierten Karten soll paarweise beginnend mit dem Zeichen `'a'` nach oben gezählt werden, sodass die ersten beiden Instanzen das Symbol `'a'`, die dritte und vierte Instanz `'b'` haben, usw. Implementieren Sie zudem die Objektmethoden `deckeAuf()`, `istAufgedeckt()` und `gibSymbol()` mit passenden Rückgabewerten.

(b) Eine Klasse `Spielfeld` mit einem zweidimensionalen Array `karten` als privates Objektfeld. Jeder Eintrag ist ein Objekt vom Typ `Karte`. Die Spielfeldgröße ist abhängig von den Parametern `breite` und `hoehe`, die im Konstruktor der Klasse als Parameter angegeben

werden.
Hinweis: Falls Höhe und Breite ungerade sind, ist die Anzahl der Karten ungerade und das Spiel nicht spielbar. Sie dürfen davon ausgehen, dass einer der beiden Parameter gerade ist, sodass dieser Fehlerfall nicht eintritt.

Im Konstruktor soll außerdem die Initialisierung des Spielfelds in zwei Schritten erfolgen:

- Füllen des Spielfelds nach folgendem Karten-Muster (hier mit `breite = 6`):

a	a	b	b	c	c
d	d	e	e	f	...
...					

- Der zweite Schritt folgt

to be continued in den Sessions **(3+2 = 5 Punkte)**

Aufgabe 9.6
Im Projekt 'tarifrechner' (s. e-Learning) finden Sie die Klassen `Tarifrechner` und `Krankenversicherung` und den enum `Tarif`.

Der Tarifrechner stellt verschiedene Methoden zur Verfügung mithilfe derer ein Java Applet erzeugt wird. Zuerst fragt das Applet den Namen und das Alter des Benutzers in der Methode `init()` ab. Anhand der eingegeben Infos sollen die verschiedenen Tarifbeiträge der Krankenversicherung ausgegeben werden.

Vervollständigen Sie dazu zunächst die Methode `berechneBeitrag(...)` der Krankenversicherung. Erweitern Sie dann den Tarifrechner um ein Feld vom Typ `Krankenversicherung`. Dieses soll mit den eingegebenen Infos in der Methode `init()` initialisiert werden. Zuletzt muss die Methode `paint()` so angepasst werden, dass sie die drei möglichen Tarife der Krankenversicherung ausgibt.

Arbeiten Sie dazu mit geeigneten Methodenaufrufen. **(5 Punkte)**