Programmierkurs
Wintersemester 2020/21
Übungsblatt 06 (14.12.2020)

M.Sc. Laslo Hunhold
Dept. Math./Inf., Abt. Informatik
Universität zu Köln

Abgabe für Rückmeldungen bis zum 21.12.2020 (08:00 Uhr), Besprechung ab dem 21.12.2020.

Aufgabe 06-0 (Zweidimensionales Array)

Betrachten Sie folgendes Listing.

```
int[][] a = { { 2, 3 }, { 3, 4, 5 }, { 6 }, { 0, 5, 5, 6 }, { 3, 2 } };
```

0. Geben Sie an, wie viele Zeilen der Array `a` hat und aus wie vielen Zellen die Zeilen jeweils bestehen.
1. Betrachten Sie die Ausdrücke `a[2][3]`, `a[3][2]`, `a[4][0]` und `a[0][0]`. Welche überschreiten die Grenzen des Arrays, und wenn nicht, welche Wert erhält man?
2. Geben Sie Ausdrücke an, mit denen man jeweils auf die erste und die zweite 6, die letzte 2 und die zweite 3 im Array zugreifen kann.

Aufgabe 06-1 (Bruchklasse)

Betrachten Sie folgendes lückenhafte Listing.

```
1  class Bruch {
2      int zaehler;
3      int nenner;
4
5      Bruch() {
6          zaehler = 0;
7          nenner = 1;
8      }
9
10     Bruch(int z, int n) {
11         zaehler = z;
12         nenner = n;
13         standardisiere();
14     }
15
16     void print() {
17         System.out.println(zaehler + "/" + nenner);
18     }
19
20     boolean gueltig() { ... }
21     void standardisiere() { ... }
22
23     void multipliziere(int f) { ... }
24     void multipliziere(Bruch f) { ... }
25
26     boolean groesser_als(int f) { ... }
27     boolean groesser_als(Bruch f) { ... }
28 }
```

Vervollständigen Sie die Methodenrümpfe und testen Sie die Klasse entsprechend (mit einer sinnvollen Kommandozeilenausgabe) in Ihrer `main`-Methode.
Die Methode `gueltig()` prüft, ob es sich bei der aktuellen Instanz um einen gültigen Bruch (i.e. der Nenner ist nicht 0) handelt. Mit der Methode `standardisiere()` wird der Bruch in eine „Standardform“ gebracht, die wir so festlegen, daß nur der Zähler negativ sein darf. Dies bedeutet also, daß wenn Zähler und Nenner oder nur der Nenner negativ sind, bei beiden eine Vorzeichenumkehr vorgenommen wird. Die überladene Methode `multipliziere()` multipliziert den Bruch mit dem Parameter, die überladene Methode `groesser_als()` prüft, ob der Wert des Bruchs den Wert des Parameters überschreitet.

Präsenzaufgabe 06-2 (Postpaketklasse)

Betrachten Sie folgendes lückenhafte Listing.

```
1  class Paket {
2      int breite, hoehe, laenge;
3      int gewicht;
4      boolean klein;
5      boolean leicht;
6      boolean mini;
7
8      Paket(int a, int b, int c, int g) { ... }
9
10     void print() {
11         System.out.println(breite + "x" + hoehe + "x" + laenge + "mm, Gewicht " +
12                 gewicht + "g, klein=" + klein + ", leicht=" + leicht +
13                 ", mini=" + mini);
14     }
15
16     boolean ist_klein() { ... }
17     boolean ist_leicht() { ... }
18     boolean ist_mini() { ... }
19 }
```

Diese Klasse beschreibt ein Postpaket, welches beispielsweise bei einem Paketdienstleister entgegengenommen und in das interne System eingetragen wird. Die einzutragenden Größen sind die Maße (Breite `breite`, Höhe `hoehe` und Länge `laenge` in mm) und das Gewicht `gewicht` (in g) des Pakets.
Beim Instanziieren der Klasse werden dem Konstruktor `Paket()` vier Parameter gegeben. Die ersten drei sind die Seitenlängen `a`, `b` und `c` (in mm) des Pakets, die *nach Länge absteigend sortiert* Breite, Höhe und Länge des Pakets sind und entsprechend in die Variablen `breite`, `hoehe` und `laenge` geschrieben werden sollen. Der vierte Parameter `g` ist das Gewicht des Pakets (in g), welches entsprechend gesetzt werden soll.
Der Paketdienstleister hat drei Klassifizierungen für Pakete: „Klein“ (ausgedrückt mit `klein`; Pakete mit Abmessungen (Breite mal Höhe mal Länge) kleiner oder gleich 600x300x150mm), „Leicht“ (ausgedrückt mit `leicht`; Pakete mit Gewicht kleiner oder gleich 2kg) und „Mini“ (ausgedrückt mit `mini`; Pakete die „klein“ und „leicht“ sind). Die Klassifizierungsmethoden `ist_klein()`, `ist_leicht()` und `ist_mini()` prüfen diese Bedingungen und geben bei Erfüllung `true` und sonst `false` zurück.
Vervollständigen Sie die Methodenrümpfe und setzen Sie alle Variablen der Klasse im Konstruktor unter Verwendung der Klassifizierungsmethoden. Testen Sie die Klasse entsprechend (mit einer sinnvollen Kommandozeilenausgabe) in Ihrer `main`-Methode.

Präsenzaufgabe 06-3 (Bruchkürzung)

Fügen Sie Ihrer vervollständigten Klasse `Bruch` aus Aufgabe 06-1 die Klassenmethode „`static int ggT(int a, int b) { ... }`“ hinzu, die den größten gemeinsamen Teiler von `a` und `b` berechnen soll. Nutzen Sie hierzu den EUKLIDischen Algorithmus:
Seien a und b *positive* ganze Zahlen mit $a > b$. Wir führen sukzessive Divsionen mit Rest durch, wobei wir in jedem folgenden Schritt den Divisor und Rest des vorherigen Schritts als Eingangswerte nehmen und aufhören, wenn der Rest 0 ist. Dann gilt, daß der Divisor genau der größte gemeinsame Teiler von a und b ist. Zum Beispiel für $a = 96$ und $b = 27$ gilt

$$\begin{aligned} 96 &= 3 \cdot 27 + 15 \\ 27 &= 1 \cdot 15 + 12 \\ 15 &= 1 \cdot 12 + 3 \\ 12 &= 4 \cdot 3 + 0, \end{aligned}$$

womit folglich der letzte Divisor 3 der größte gemeinsame Teiler von 96 und 27 ist.
Fügen Sie Ihrer Klasse eine Methode „`void kuerze() { ... }`“ hinzu, die den Bruch unter Verwendung der Methode `ggT()` kürzt. Achten Sie auf eventuelle negative Vorzeichen und die Möglichkeit, daß der Zähler 0 sein kann, wofür der EUKLIDische Algorithmus nicht definiert ist. Testen Sie Ihre `kuerzen()`-Methode (mit einer sinnvollen Kommandozeilenausgabe) in Ihrer `main`-Methode.