TU Ilmenau, Fakultät IA
FG Telematik/Rechnernetze
Prof. Dr.-Ing. G. Schäfer
http://www.tu-ilmenau.de/telematik/pua

TECHNISCHE UNIVERSITÄT
ILMENAU

Programmierung und Algorithmen WS 23/24
Übungsblatt 11

Die Lösungen der Aufgaben sind bis zum 21.01.24, 23:59 Uhr abzugeben.

Die Besprechung der Aufgaben erfolgt in KW 4.

Aufgabe 1 (Implementierung komplexer Datentypen) **6 Punkte**

Implementieren Sie eine Java-Klasse `NatList` welche eine Liste mit natürlichen Zahlen repräsentiert. Verwenden Sie dabei *keine* anderen Klassen aus der Java-Klassenbibliothek, mit Ausnahme von Arrays (d.h. Arrays dürfen Sie natürlich verwenden)! Zur Repräsentation natürlicher Zahlen können Sie in Java `int` verwenden. Zur Speicherung der eigentlichen Listenelemente in Ihrer Klasse können Sie beispielsweise ein Feld vom Typ `int[]` verwenden. Die Klasse soll die folgenden Methoden haben:

- Ein Konstruktor soll ein Objekt Ihrer Klasse als leere Liste initialisieren.
- `head()`: Gibt das erste Element der Liste zurück.
- `tail()`: Gibt eine Liste der hinteren $n-1$ Elemente einer n-elementigen Liste zurück (als Typ `NatList`).
- `append(int x)`: Fügt ein Element hinten an die Liste an.
- `length()`: Gibt die Länge der Liste zurück.
- `isSorted()`: Liefert genau dann `true`, wenn die Liste aufsteigend sortiert ist.

Aufgabe 2 (Datenstruktur für rationale Zahlen) **11 Punkte**

Auf den Folien 27 bis 31 des Foliensatzes "Objektorientierung" wird eine Klasse `RatNumber` zur Repräsentation rationaler Zahlen vorgestellt. Das entsprechende Grundgerüst finden Sie auch in der Datei `RatNumberProg.java` in der WebIDE und im Moodle-Kurs. Erweitern Sie die Klasse, so dass ihre Datenstruktur die folgenden Operationen unterstützt:

(a) Einen zusätzlichen Konstruktor, welcher eine Ganzzahl übergeben bekommt und damit das `RatNumber`-Objekt entsprechend initialisiert.

(b) `normalize()`: Kürzen der rationalen Zahl auf eine Repräsentation mit minimalem Zähler und Nenner.

(c) `toString()`: Ausgabe als String (in Form eines Bruches)

(d) `isZero()`: Test auf Gleichheit zum Wert Null.

(e) `equals(RatNumber other)`: Test auf Gleichheit mit zweiter rationaler Zahl.

(f) `mult(..)`, `div(..)`, `minus(..)`: Implementierung der Operationen Multiplikation, Division und Subtraktion

Achten Sie auf die Wahl geeigneter Rückgabetypen und Sichtbarkeit der Methoden.

Aufgabe 3 (Overriding und Overloading) **4 + 1 Punkte**

Das unten stehende Java-Programm nutzt die Konzepte Overriding und Overloading.

(a) Die mit `/** A **/` gekennzeichnete Stelle wird mehrmals durchlaufen. Nehmen Sie an, die Indexvariable `i` der umgebenden Schleife habe den Wert 3.

Geben Sie an, welche Methoden bei der Auswertung dieser Anweisung mit welchen Parametern aufgerufen werden. Kennzeichnen Sie für jede aufgerufene Methode, in welcher Klasse sie definiert wurde. Welcher Wert wird schließlich zu `sum` addiert?

(b) Wie ist die Ausgabe des Programms?

```
1  class X {
2    int x;
3    public X(int x) { this.x = x; }
4    public int a(int x, int y) { return x * y; }
5    public int a(int x) { return a(x, this.x); }
6  }
7
8  class Y extends X {
9    public Y(int x) { super(x); } // Konstr. d. Oberkl.
10   public int a(int x, int y) { return a(x * y, x, y); }
11   public int a(int x, int y, int z) { return x * y + z; }
12 }
13
14 class Prog {
15   public static void main(String[] args) {
16     X[] x = new X[4];
17     for(int i = 0; i < x.length; ++i) {
18       x[i] = new Y(i);
19     }
20     int sum = 0;
21     for(int i = 0; i < x.length; ++i) {
22       sum += x[i].a(i); /** A **/
23     }
24     System.out.println(sum);
25   }
26 }
```

Aufgabe 4 (Große Zahlen in Java) **1 + 1 + 4 + 5 Punkte**

Für die Speicherung und Verarbeitung von großen Zahlen stellt Ihnen Java einige primitive Datentypen (z.B. `long`, `double`) zur Verfügung.

(a) Welches Problem kann auftreten, wenn Sie versuchen sehr große Zahlen in eine Variable vom Typ `double` zu speichern?

(b) Welches Problem kann auftreten, wenn Sie versuchen sehr große Zahlen in eine Variable vom Typ `long` zu speichern?

(c) Implementieren Sie eine Java-Klasse `BigNum`, welche eine beliebig große *natürliche Zahl* aufnehmen kann, so dass keine der Probleme aus (a) und (b) auftreten. Nutzen Sie dazu intern ein Array aus Ziffern. Implementieren Sie mindestens einen Konstruktor, welcher eine als `String` gegebene Zahl in Dezimaldarstellung einliest. Implementieren Sie zusätzlich eine Methode `toString`, welche die Zahl wieder in einen String umwandelt und diesen zurückgibt.

(d) Erweitern Sie die Klasse `BigNum` um die Methoden `add` und `mult`, welche jeweils ein Objekt vom Typ `BigNum` als Argument erhalten und ein neues Objekt vom Typ `BigNum` zurückgeben.