Programmierpraktikum

Wintersemester 2021/22

Prof. Dr. rer. nat. habil. Petra Hofstedt
Ilja Becker M. Sc., Sven Löffler M. Sc.
Sonja Breuß B. Sc., Carlo Bückert B. Sc., Endris Wenzke

Brandenburgische Technische Universität Cottbus - Senftenberg

Übungsblatt 7

Abgabedatum: 05.12.2021

Die Abgabe Ihrer Lösungen erfolgt vor Ablauf der Abgabefrist digital über die Moodle-Plattform. Erstellen Sie dazu ein PDF-Dokument, das die Lösungen Ihrer schriftlichen Aufgaben enthält. Laden Sie dieses PDF-Dokument und den erarbeiteten Java-Code (.java-Dateien) mit den in den Aufgaben vorgegebenen Namen bei Moodle hoch. Bitte laden Sie die Dateien einzeln hoch, Dateiarchive (z.B. .zip-Dateien) werden nicht akzeptiert.

Sie können maximal **(8 Punkte)** mit diesem Übungsblatt erreichen.

Aufgabe 1 (Fibonacci-Zahlen) — 2 Punkte

Im 13. Jahrhundert beschrieb der italienische Mathematiker Leonardo da Pisa alias Fibonacci eine Zahlenfolge, die später als Fibonacci-Folge bekannt wurde. Fibonacci illustrierte die Folge mit dem Wachstum einer Kaninchenpopulation. Die Population startet im ersten Monat mit einem neugeborenen Kaninchenpaar. Jedes Kaninchenpaar ist nach zwei Monaten geschlechtsreif und wirft ab diesem Zeitpunkt im Monatsabstand je ein neues Paar. Das i-te Glied der Fibonacci-Folge, welches wir f_i nennen, gibt die Anzahl der Kaninchenpaare im i-ten Monat an, falls bis dahin kein Kaninchen gestorben ist.

Die Fibonacci-Folge kann folgendermaßen formal definiert werden:

$$\begin{aligned} f_1 &= 1 \\ f_2 &= 1 \\ f_{i+2} &= f_i + f_{i+1} \end{aligned}$$

Dabei gilt die letzt Gleichung für alle positiven ganzen Zahlen i.

Die erste Gleichung kennzeichnet den Ausgangszustand mit einem Kaninchenpaar. Im zweiten Monat ist dieses Paar noch nicht geschlechtsreif, sodass f_2 auch 1 ist, wie in der zweiten Gleichung definiert. Ab dem dritten Monat werden dann Kaninchen geworfen. Die geschlechtsreifen Paare sind immer gerade diejenigen, die vor zwei Monaten schon zur Population gehörten. Es sei i die Nummer des vorletzten Monats. Dann ist f_i die Anzahl der geschlechtsreifen Paare. Da jedes dieser Paare genau ein neues Paar wirft, werden also f_i Paare geworfen. Diese kommen zu den Paaren hinzu, die schon im letzten Monat zur Population gehörten. Deren Anzahl ist f_{i+1}. Also gibt es im betrachteten Monat insgesamt $f_{i+2} = f_i + f_{i+1}$ Kaninchenpaare, was durch die dritte Gleichung beschrieben wird.

Legen Sie eine neue `Klasse Fibonacci.java` an.

1. Schreiben Sie eine Methode, die für eine gegebene Zahl i das entsprechende Folgenglied f_i ermittelt. Verwenden Sie eine rekursive Implementierung, die unmittelbar die obige Definition der Fibonacci-Folge wiederspiegelt. Ihre Methode soll folgende Signatur besitzen:

```
public static long fibonacciRekursiv(int index)
```

2. Schreiben Sie eine Methode `fibonacciIterativ`, die zu einer Zahl i die Fibonacci- Zahl f_i ohne Verwendung von Rekursion ausrechnet. Die Methode soll mittels einer Schleife nacheinander alle Fibonacci-Zahlen bis zu dem gewünschten Folgenglied berechnen. Dabei soll jedes Folgenglied f_{i+2} durch Addition der vorher berechneten Glieder f_i und f_{i+1} ermittelt werden. Verwenden Sie für `fibonacciIterativ` eine Signatur analog zu der von `fibonacciRekursiv`.

3. Die Glieder der Fibonacci-Folge können auch mit der Formel von Moivre und Binet berechnet werden:

$$f_n = \frac{1}{\sqrt{5}}[(\frac{1+\sqrt{5}}{2})^n - (\frac{1-\sqrt{5}]}{2})^n]$$

   Schreiben Sie eine Methode `fibonacciFormel`, welche Fibonacci-Zahlen mittels der Formel von Moivre und Binet berechnet. Die Signatur von `fibonacciFormel` soll ebenfalls der Signatur von `fibonacciRekursiv` entsprechen.

4. Schreiben Sie eine `main`-Methode, die einen Index von der Tastatur einliest und anschließend jede der drei oben beschriebenen Methoden auf diesen Index anwendet und nach Beendigung der jeweiligen Methode sofort deren Ergebnis ausgibt. Fügen Sie hinter jedem entsprechenden Aufruf von `System.out.println` die Anweisung `System.out.flush();` ein, weil sonst nicht garantiert werden kann, dass die Ausgaben sofort erscheinen.

5. Führen Sie das Programm mit den Eingabewerten 5, 15, 25, 35, 40 und 47 aus und vergleichen Sie die Laufzeiten und die Ergebnisse der drei Funktionen. Beschreiben Sie Ihre Entdeckungen. und benennen Sie Vor- und Nachteile der drei Implementierungen.

Aufgabe 2 (Euklidische Algorithmus) 1 Punkt

Implementieren Sie den Euklidischen Algorithmus in einer Klasse `Euklid.java` rekursiv. Verwenden Sie ausser Rekursion nur if-else-Anweisungen, Vergleiche und Subtraktion.

Der Euklidische Algorithmus zur Berechnung des größten gemeinsamen Teilers zweier positiver ganzer Zahlen a und b (ggt(a,b)) ist wie folgt rekursiv definiert:

ggt(a,b)	= a,	falls a = b gilt
ggt(a,b)	= ggt(a - b, b),	falls a > b gilt
ggt(a,b)	= ggt(a, b - a),	falls b > a gilt

Aufgabe 3 (Palindrom und Raute rekursiv) 2 Punkte

Lösen Sie die Palindrom-Aufgabe und die Raute-Aufgabe von Aufgabenblatt 3 (Aufgabe 2.1 und Aufgabe 2.3) rekursiv, ohne Verwendung von Schleifen. Die Aufgaben dort lauteten:

> Schreiben Sie ein Programm (Palindrom.java), das einen Text einliest und ausgibt ob dieser ein Palindrom ist. Ein Palindrom ist eine Folge, die identisch mit ihrer Umkehrung ist.
>
> Schreiben Sie eine Anwendung `Raute.java`, welche ein Rautenmuster aus # ausgibt. Der Nutzer soll zuvor die Breite *b* der Raute eingeben können. Bevor eine Raute ausgegeben wird, soll getestet werden ob *b* eine gerade Zahl ist, ist dass der Fall, so soll das Programm mit der

Textausgabe „Die Raute kann nur für ungerade Eingaben gezeichnet werden."ausgegeben werden.

Verwenden Sie für das lösen der Aufgaben **KEINE** Schleifen, sondern nur Rekursion.

Aufgabe 4 (Brüche) 3 Punkte

Mit Hilfe des Entwurfsparadigmas der Objektorientierung kann eine Vielzahl von Dingen modellhaft nachgebildet werden. Je komplexer die nachzubildende Struktur, desto komplexer auch das korrespondierende Modell. So ein Modell wird durch eine sogenannte Klasse repräsentiert, welche als eine Art Schablone dient, mit der Objekte (auch Instanzen genannt) mit gleichen Attributen und Methoden erzeugt (oder auch instanziiert) werden können. Es lassen sich auf diese Weise leicht neue, zusammengesetzte Datentypen erzeugen. Ein Beispiel dafür wäre die Modellierung eines Bruchs $\frac{p}{q} \in \mathbb{Q}$ mit $p, q \in \mathbb{Z}$.

1. Schreiben Sie eine Klasse `Bruch.java`, die einen Bruch ganzer Zahlen modelliert. Ein solcher besteht aus einem ganzzahligen Zähler und Nenner. Zähler und Nenner sollen als `private` deklariert werden. Aus anderen Klassen soll auf diese mittels sogenannter `Getter`-Methoden zugegriffen werden können.
2. Schreiben Sie zwei Konstruktoren für die Klasse Bruch. Der eine Konstruktor soll einen Nenner und einen Zähler übergeben bekommen und die Objektattribute dem entsprechend setzen. Der andere Konstruktor soll nur eine ganze Zahl als Eingabe erhalten und mittels Aufruf des anderen Konstruktors diese Zahl als Bruch darstellen. Überlegen Sie sich dazu einen passenden Wert für den Nenner.

   Bei dem Konstruktor mit der Eingabe von Zähler und Nenner soll überprüft werden, ob der Nenner ungleich Null ist. Sollte er Null sein, so soll das Programm beendet werden.
3. Überschreiben Sie die `toString`-Methode derart, dass sie den Bruch in der Form „(≪ *Wert vom Zähler* ≫ / ≪ *Wert vom Nenner* ≫)" zurück gibt.
4. Schreiben Sie statische Methoden für die vier Grundrechenarten auf Brüchen (Addition, Subtraktion, Multiplikation, Division).
5. Schreiben Sie drei nicht statische Methoden, wobei die eine den Dezimalwert des Bruches wiedergibt, eine das Reziproke eines Bruchs zurückgibt und die andere den Bruch soweit wie möglich kürzt.
6. Legen Sie eine separate Klasse (`Main.java`) an, und erzeugen Sie in deren `main`-Methode, mit Hilfe der beiden Konstruktoren, drei Objekte der Klasse Bruch mit folgenden Werten:

   bruch1 = $\frac{1}{5}$, bruch2 = 3, bruch3 = $\frac{5}{2}$

   Führen Sie des Weiteren folgende Berechnungen aus und lassen Sie sich die Ergebnisse normal, gekürzt und als Dezimalzahl ausgeben:

   result1 = bruch1 + bruch2 - bruch3
   result2 = result1 * bruch2 / bruch3