Brandenburgische
Technische Universität
Cottbus - Senftenberg

Brandenburgische Technische Universität Cottbus – Senftenberg
Institut für Informatik
PD Dr.-Ing. habil. Thomas Hinze

Sommersemester 2016

Abgabe der Lösungen: bis Mittwoch, 15.06.2016

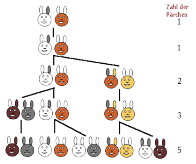
Einführung in die Programmierung
Übungsblatt 4

Hinweise

- Erstellen Sie bitte eine schriftliche Lösung der nachfolgenden Aufgaben und erzeugen Sie daraus eine pdf-Datei. Sie können auch Ihre (lesbare!) handschriftliche Lösung als pdf einscannen. Achten Sie bitte darauf, dass am Beginn der ersten Seite des pdfs Ihr *Name*, Ihre *Matrikelnummer* und der Name Ihres *Tutors* (Laborübungsgruppe) vermerkt ist. Jede Aufgabe, die durch **(Laborübung)** gekennzeichnet ist, beinhaltet als Lösung auch ein Java-Programm als Quelltextdatei. Der Quelltextdateiname muss dem Namen der *Klasse* entsprechen, die die `main`-Methode enthält. Bitte packen Sie Ihr pdf gemeinsam mit allen Java-Quelltextdateien (nur `*.java`, keine `*.class`) in eine zip-Datei, die Sie termingemäß im moodle hochladen.
- Sie bekommen eine Bewertung Ihrer Lösung mit der Information, ob sie ausreichend zum Bestehen des Übungsblattes ist. Im Falle einer nichtausreichenden Lösung haben Sie Gelegenheit, eine verbesserte Version zeitnah, aber spätestens bis 15.07.2016, im moodle nachzureichen. Plagiate werden nicht als ausreichende Lösungen anerkannt.

Aufgabe 1

Ein Kaninchenpaar wird in einem vollständig abgegrenzten Gebiet ausgesetzt. Von Natur aus zeugt jedes Kaninchenpaar ab dem zweiten Lebensmonat ein weiteres Paar pro Monat. Dieses wiederum beginnt vom zweiten Lebensmonat an, sich auf gleiche Weise fortzupflanzen (siehe Grafik). Wir nehmen an, dass keine Kaninchen sterben.



www.wikipedia.de

- Entwickeln Sie eine rekursive Bildungsvorschrift, aus der sich die Anzahl Kaninchenpaare $\mathrm{f}(n)$ nach n Monaten berechnen lässt.
- Recherchieren Sie, unter welchem vordefinierten Namen die rekursive Bildungsvorschrift bereits bekannt ist.
- Schätzen Sie den Werteverlauf Ihrer rekursiven Funktion nach oben ab.
- Wie lässt sich Ihre rekursive Bildungsvorschrift ohne Rekursion in Java implementieren?
- **(Laborübung)** Schreiben Sie sowohl ein Java-Programm für die rekursive als auch ein Java-Programm für eine nicht-rekursive Berechnung und vergleichen Sie anhand von selbstgewählten Fallstudien, wieviele Berechnungsschritte (Anzahl rekursive Aufrufe einerseits und Anzahl arithmetische Operationen andererseits) jeweils nötig sind.

Aufgabe 2

Die *Quersumme* einer natürlichen Zahl im Dezimalsystem bietet eine elegante Möglichkeit, die Teilbarkeit durch 3 und 9 zu testen. Darüber hinaus finden Quersummen als Prüfsummen in fehlererkennenden bzw. fehlerkorrigierenden Übertragungscodes Verwendung. Die Quersumme einer natürlichen Zahl ist definiert als Summe ihrer einzelnen Dezimalziffern, beispielsweise besitzt die Zahl 582 die Quersumme 15. Eine rekursive Berechnungsvorschrift ist gegeben durch:

$$\text{quersumme}(n) = \begin{cases} n & \text{falls} \quad n \leq 9 \\ \text{quersumme}(\lfloor \frac{n}{10} \rfloor) + (n \mod 10) & \text{sonst} \end{cases}$$

- **(Laborübung)** Implementieren Sie diese rekursive Berechnungsvorschrift in Java und testen Sie sie anhand mehrerer Fallstudien aus.
- Ist der Speicherplatzbedarf zur Ausführung des rekursiven Java-Programmes abhängig vom eingegebenen Wert für n? Begründen Sie bitte Ihre Entscheidung.
- **(Laborübung)** Entwickeln Sie ein Java-Programm zur Quersummenberechnung, das ohne Rekursion auskommt und vergleichen Sie verbal seinen Speicherplatzbedarf mit der rekursiven Variante.

Aufgabe 3

Rationale Zahlen („Brüche“) finden in vielfältigen numerischen Algorithmen und Computersimulationen Verwendung. Nicht selten besteht dabei die Anforderung, das Rechnen mit Brüchen präzise und ohne mögliche Ungenauigkeiten bei Verwendung von Gleitkommaoperationen auszuführen. Ziel dieser Aufgabe ist es, eine eigene Klasse zu entwickeln, die Werkzeuge zum Bruchrechnen bereitstellt. Ein Objekt (ein Exemplar) dieser Klasse soll in der Lage sein, einen konkreten Bruch als Attribut zu speichern, indem Zähler und Nenner jeweils als ganze Zahlen abgelegt werden, wobei der Bruch dabei automatisch soweit wie möglich gekürzt wird mithilfe des *Euklidischen Algorithmus*
(`http://de.wikipedia.org/wiki/Euklidischer_Algorithmus`).
Die Methoden der Klasse sollen es ermöglichen, auf den als Attribut gespeicherten Bruch einen anderen Bruch zu addieren, einen anderen Bruch zu subtrahieren, zu multiplizieren, zu dividieren und als Zeichenkette für eine Ausgabe bereitzustellen.

- **(Laborübung)** Schreiben Sie eine Klasse `Bruch`, die einen Bruch ganzer Zahlen modelliert. Ein solcher besteht aus einem ganzzahligen Zähler und Nenner. Bedenken Sie, dass der Nenner 0 nicht zulässig ist und in einem solchen Fall kein Bruch definiert wird. Das Anlegen eines Bruches geschieht über den *Konstruktor*, der bei Erzeugen eines neuen Objektes der Bruchklasse aufgerufen wird.
- **(Laborübung)** Fügen Sie Zugriffsmethoden hinzu, welche den lesenden Zugriff auf Zähler und Nenner ermöglichen, also eine Methode `gibZaehler`, die den als Attribut gespeicherten Zähler zurückliefert und eine weitere Methode `gibNenner`, die den Nenner zurückliefert.
- **(Laborübung)** Implementieren Sie Methoden für *Addition*, *Subtraktion*, *Multiplikation*, *Division* sowie `toString` für die Umwandlung des Bruches in eine Zeichenkette. Verleihen Sie den Methoden treffende Bezeicher.
- **(Laborübung)** Eine übliche Verfahrensweise bei der Verwendung von Brüchen ist das Kürzen. Es ist dazu hilfreich , den größten gemeinsamen Teiler (engl. *greatest common divisor*, kurz *gcd*) von Zähler und Nenner zu ermitteln. Verwenden Sie hierfür den *Euklidischen Algorithmus*. Implementieren Sie ihn am besten in einer eigenen Methode; diese sollte jedoch nicht öffentlich sein. Schreiben Sie nun eine Methode `kuerzen`, die den Bruch des Objekts, auf dem die Methode aufgerufen wurde, kürzt,

sofern dies auch möglich ist. Zusätzlich normiert diese Methode das Vorzeichen des Bruches so, dass bei negativen Brüchen das Vorzeichen „−“ stets im Zähler steht und bei nichtnegativen Brüchen keine Minuszeichen auftreten.

- **(Laborübung)** Schreiben Sie zum Testen der Klasse `Bruch` eine gesonderte Testklasse in die gleiche java-Datei, die als einzige Methode `main` enthält und darin verschiedene Brüche anlegt, arithmetisch miteinander verknüpft und die Ergebnisse ausgibt. Ein Beispiel für die Testklasse könnte wie folgt aussehen:

```
public class MeinBruchrechnungsTest {
  public static void main(String[] args) {
    Bruch x = new Bruch(2, 3); // Bruch 2/3 mittels Konstruktor anlegen
    System.out.println(x.toString()); // Ausgabe des Bruches
    Bruch y = new Bruch(1, 2); // Bruch 1/2 mittels Konstruktor anlegen
    x.addiere(y);
    System.out.println(x.toString()); // Ausgabe der Summe 2/3 + 1/2 = 7/6
    Bruch z = new Bruch(4); // Bruch 4/1 mittels weiterem Konstruktor anlegen
    x.multipliziere(z);
    System.out.println(x.toString()); // Ausgabe des Produkts 4 * 7/6 = 14/3
    Bruch v = new Bruch(18, -24); // Bruch -3/4 mittels Konstruktor anlegen
    x.subtrahiere(v);
    System.out.println(x.toString()); // Ausgabe der Differenz 14/3 - (-3/4) = 65/12
    Bruch w = new Bruch(-5,-3); // Bruch 5/3 mittels Konstruktor anlegen
    x.dividiere(w);
    System.out.println(x.toString()); // Ausgabe des Quotienten (65/12) / (5/3) = 13/4
  }
}
```

Aufgabe 4

Auf der Lehrveranstaltungswebseite finden Sie die herunterladbare Datei `cars.zip`. Nach dem Entpacken erhalten Sie daraus drei Java-Dateien (`Car.java`, `RacingCar.java` und `Main.java`), wobei jede davon eine entsprechende Java-Klasse beinhaltet. In der Klasse `Car` sind Attribute und Methoden enthalten, die ausgewählte Eigenschaften und Aktionen eines Fahrzeugs nachbilden. Die Klasse `RacingCar` erbt von `Car` und spezialisiert dabei auf Rennfahrzeuge.

- Die in der Klasse `RacingCar` definierten Attribute sind als `final` deklariert. Was bedeutet das und welche Konsequenzen ergeben sich daraus?
- Auf welche Attribute (eigene und von `Car` ererbte) kann innerhalb der Klasse `RacingCar` direkt zugegriffen werden?
- Welche Methoden (eigene und von `Car` ererbte) stehen in der Klasse `RacingCar` zur Verfügung?
- Welche aus `Car` ererbten Methoden werden in `RacingCar` *überschrieben*?
- **(Laborübung)** Testen Sie den Zugriff auf Attribute und Methoden der Klassen `Car` und `RacingCar`, indem Sie in der Klasse `Main` entsprechende Objekte anlegen und damit in selbstgewählten Experimentierszenarien arbeiten.