MINT-KOLLEG BADEN-WÜRTTEMBERG

Universität Stuttgart

Vorkurs Informatik

Dr. Sebastian Müller
Dr. Frank Schweiner

---

3. Tutoriumsblatt

Bearbeiten Sie dieses Übungsblatt im Projekt `Day3` und legen Sie für jede Aufgabe ein Paket `task01`, `task02`, ... an.

Aufgaben Vormittag

1. Aufgabe

Taschenrechner / Calculator

Teil 1: Funktionen

a) Legen Sie eine Klasse `Calculator` an und erstellen Sie in dieser die folgenden Methoden. Achten Sie genau auf die Datentypen.

(a) `int add(int a, int b)`
berechnet die Summe von a und b.

(b) `double divide(int a, int b)`
berechnet a/b mit Nachkommastellen.

(c) `int max(int n, int m)`
gibt die größere der beiden Zahlen zurück.

(d) `int max(int x, int y, int z)`
gibt die größte der drei Zahlen zurück und nutzt dabei `int max(int n, int m)`.

(e) `void rectangle(int height, int width)`
gibt ein Rechteck aus Sternen mit den gegebenen Kantenlängen aus. Beispiel für `2,3`:

```
***
***
```

(f) `void square(int length)`
gibt ein Quadrat aus Sternen mit der gegebenen Kantenlänge aus und nutzt dabei `void rectangle(int height, int width)`

(g) `boolean isPrime(int n)`
überprüft, ob `n` eine Primzahl ist.

b) Testen Sie die Methoden. Legen Sie dazu eine weitere Klasse `Test` an, die die `main`-Funktion enthält. Erzeugen Sie einen neuen Taschenrechner `Calculator calc = new Calculator();` und rufen Sie auf diesem die Methoden auf, z.B. `calc.divide(5,6);`

Bitte wenden!

Teil 2: Heron-Verfahren

a) Informieren Sie sich über das Heron-Verfahren zur Berechnung der Quadratwurzel einer beliebigen positiven Zahl a (z.B. bei Wikipedia).

b) Implementieren Sie das Verfahren in einer eigenen Methode in der Klasse `Calculator`. Lassen Sie das Ergebnis nach 3 Iterationen zurückgeben.

c) Lassen Sie sich in der `main`-Funktion die Quadratwurzeln aller Zahlen zwischen 1 und 20 berechnen und geben Sie sie auf der Konsole aus.

d) Vergleichen Sie Ihre Ergebnisse mit den Werten, die `Math.sqrt` liefert.

e) *Zusatzaufgabe*: Offensichtlich reichen 3 Iterationen nicht aus, um perfekte Ergebnisse zu erhalten. Schreiben Sie eine weitere Methode `heronExact`, in der mit einer `while`-Schleife so lange gerechnet wird, bis sich das Ergebnis nicht mehr ändert. Erst dann soll es zurückgegeben werden.

Aufgaben Nachmittag

2. Aufgabe

Taschenrechner / Calculator

Teil 3: Exceptions

a) Die Operationen `add` und `divide` des Taschenrechners funktionieren nicht für alle Eingaben fehlerfrei. Bei `add` kann es zu Integer-Overflows bzw. -Underflows kommen, bei `divide` könnte durch 0 geteilt werden.
Ergänzen Sie beide Operationen so, dass `IllegalArgumentExceptions` geworfen werden, falls die Argumente bei Methodenaufruf nicht korrekt sind.
Hinweis: Verwenden Sie den Datentyp `long`, um festzustellen, ab das Ergebnis von `add` zu groß oder zu klein wird.

b) Rufen Sie die Operationen mit fehlerhaften Eingaben in der `main`-Methode auf. Was passiert?

Bitte wenden!

3. Aufgabe

Input

a) Ein `Scanner` erlaubt es, Zahlen von der Konsole einzulesen, während das Programm läuft. Legen Sie eine Klasse `Input` an und kopieren Sie folgenden Code in Ihre `main`-Methode:

```
Scanner scanner = new Scanner(System.in);
int input = 0;
System.out.println("Please insert an integer number:");
input = scanner.nextInt();
System.out.println("Your input was: " + input);
```

Testen Sie dieses Programm.

b) Die Methode `nextInt()` wirft eine `InputMismatchException`, falls die Eingabe keine ganze Zahl ist (also z.B. `"eins"` oder `2.3`). Fangen Sie diese auf und brechen Sie das Programm ab mit der Nachricht `"Wrong Input!"`.

c) *Zusatzaufgabe:* Fordern Sie bei falschen Eingaben den Benutzer so lange auf, eine ganze Zahl einzugeben, bis die Eingabe tatsächlich eine ganze Zahl ist. Achtung: Bei Verwendung von `nextInt()` müssen Sie in der Fehlerbehandlung erst die falsche Eingabe mit `nextLine()` überspringen bevor Sie weiter einlesen.

4. Aufgabe

Rekursion / Recursive Calculator

Legen Sie eine Klasse `RekursiveCalculator` an und erstellen Sie in dieser die folgenden Methoden.

Teil 1: Mystery

Gegeben sei die folgende rekursive Definition von `mystery`:

- `mystery(0, q) = q`
- `mystery(p, q) = mystery(p-1, q+1)`

a) Berechnen Sie von Hand `mystery(2,4)` und `mystery(4,3)`. Welche (einfache) mathematische Funktion wird durch `mystery` berechnet?

b) Implementieren Sie in der Klasse `RecursiveCalculator` die Methode `mystery` als rekursive Methode, die zwei Argumente vom Typ `int` erwartet und einen `int` zurückgibt.

Bitte wenden!

Teil 2: Collatz-Problem

Ein ungelöstes Problem der Mathematik betrifft die Hailstone-Zahlenreihe. Sie wird von einer positiven Zahl n ausgehend wie folgt berechnet:

- Ist n die Zahl 1, stoppe.
- Ist n gerade, konstruiere den Rest der Hailstone-Reihe ab $n/2$.
- Ist n ungerade, konstruiere den Rest der Hailstone-Reihe ab $3n + 1$.

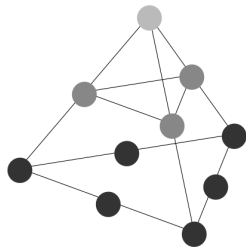
Es wird vermutet dass jede so konstruierte Zahlenfolge mit der 1 endet. Allerdings ist diese Vermutung noch nicht bewiesen. Einige Beispiele für die resultierenden Reihen mit verschiedenen Startwerten:

- Startwert 3: 3, 10, 5, 16, 8, 4, 2, 1.
- Startwert 6: 6, 3, 10, 5, 16, 8, 4, 2, 1.
- Startwert 19: 19, 58, 29, 88, 44, 22, 11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1.

Implementieren Sie eine rekursive Methode `void printHailstone(int n)`, die die Hailstone-Zahlenreihe von einem Startwert n ausgehend berechnet und die (Zwischen)Ergebnisse sofort auf der Konsole ausgibt.

Teil 3: Tetraederzahlen

Die Tetraederzahl beschreibt die Anzahl Bälle in dem Tetraeder (der Pyramide), die entsteht, indem man Dreiecke übereinanderlegt, deren Seitenlängen von oben nach unten jeweils um eins zunehmen. Dabei ist n die Anzahl der übernanderliegenden Dreiecke. In jeder Lage liegen $m = \sum_{i=1}^{n} i$ Bälle (diese Zahl haben wir in der Vorlesung als `sum(n)` berechnet).



Die Tetraederzahlen lassen sich rekursiv berechnen, indem man die Summe aller Bälle in allen Lagen der Pyramide berechnet:

- `tetrahedron( 1 ) = 1`
- `tetrahedron( n ) = tetrahedron( n-1 ) + sum( n )`

Setzen Sie diese rekursive Definition in einer Methode `int tetrahedron(int n)` um.
Die ersten Tetraederzahlen sind 1, 4, 10, 20, 35, 56, 84, 120, ...

Bitte wenden!

Zusatzaufgaben

5. Aufgabe

Rekursiv vs. Iterativ / Fibonacci

Die Fibonacci-Zahlen sind gegeben durch $0, 1, 1, 2, 3, 5, 8, 13, \ldots$. Dabei ist die nächste Fibonacci-Zahl immer die Summe der beiden Vorgänger.

a) Legen Sie eine Klasse `Fibonacci` an, in der Sie die folgenden Methoden implementieren:

- Schreiben Sie eine Methode `double getNthNumber_r(int n)`, welche die n-te Fibonacci-Zahl rekursiv berechnet.
- Schreiben Sie eine Methode `double getNthNumber_i(int n)`, welche die n-te Fibonacci-Zahl mit einer `for`-Schleife berechnet.

b) Legen Sie eine weitere Klasse `Test` mit der `main`-Methode an. Bestimmen Sie, wie lange die Berechnung der 30., 35., 40., 42., 45. Fibonacci-Zahl jeweils benötigt. Fragen Sie hierzu vor und nach jedem Methodenaufruf die Systemzeit mit `System.currentTimeMillis()` ab und lassen Sie sich die Differenz ausgeben. Welche Methode ist schneller und warum?

c) Schreiben Sie eine Methode `double calculatePhi()`, die den Grenzwert $\Phi = \lim_{n\to\infty} \frac{F(n+1)}{F(n)}$ berechnet. Informieren Sie sich anschließend bei Wikipedia über die Besonderheit des goldenen Schnittes Φ.

6. Aufgabe

Umwandlung zwischen Zahlensystemen (sehr schwer)

a) Informieren Sie sich zunächst in den Zusatzmaterialien mit dem Foliensatz *0_ Informationsdarstellung* über das Binär- und das Hexadezimalsystem.

b) Legen Sie eine Klasse `Converter` an, in der Sie die folgenden Methoden implementieren: `String decimalToBinary(int number)` und `String decimalToHex(int number)`, Diese bekommen eine Dezimalzahl als Eingabe und geben eine Binär- oder Hexadezimalzahl als String zurück.

c) Testen Sie beide Methoden, indem Sie sie in der `main`-Methode mit unterschiedlichen Argumenten aufrufen.

Hinweis: Umrechnung Dezimalzahl in Binär- / Hexadezimalzahl:

Teile die Zahl mit Rest durch die Basis (2 bzw. 16)
→ Divisionsrest ist die nächste Ziffer (von rechts nach links)
→ Teile den Quotienten weiter, so lange bis das Ergebnis 0 ist.

Beispiel: 14/2=7 Rest 0, 7/2=3 Rest 1, 3/2=1 Rest 1, 1/2=0 Rest 1 → $14_{10} = 1110_2$

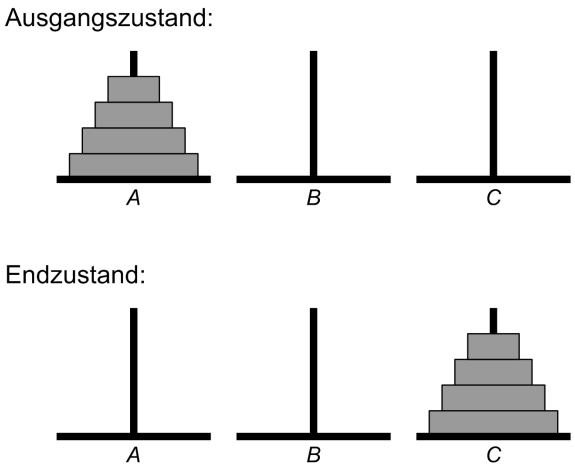
Außerdem könnte die Methode `String Integer.toString(int a)` hilfreich sein.

Bitte wenden!

7. Aufgabe

Türme von Hanoi (sehr schwer)

"Die Türme von Hanoi" ist ein mathematisches Knobelspiel, das vermutlich 1883 vom französischen Mathematiker Édouard Lucas erfunden wurde. In dem Spiel ist ein Turm aus Scheiben auf einem Pfosten aufgestapelt. Die größte Scheibe liegt unten, die kleinste oben. Dieser Turm soll nun auf einen anderen Pfosten transferiert werden. Ein dritter Pfosten darf als Zwischenablage mit benutzt werden. Die Schwierigkeit besteht nun darin, dass immer nur die oberste Scheibe eines Stapels bewegt werden darf und dass niemals eine größere Scheibe auf einer kleineren Scheibe liegen darf.



Schreiben Sie eine Methode, die die Anweisungen zur Verlagerung eines Turmes einer gegebenen Höhe von einem Pfosten zum nächsten ausgibt.

Die Grundidee ist dabei folgende: Um einen Turm der Höhe k von A nach B zu bewegen, muss zuerst ich einen Turm der Höhe $k-1$ von A nach C verschieben. Dann kann ich die eine noch übrige Scheibe von A nach B legen. Zum Schluss kann ich den $k-1$ hohen Turm von C nach B verschieben. Dieses Verfahren wird rekursiv angewendet bis Türme der Höhe 0 erreicht sind.

Schritte, um einen Turm der Höhe $k = 3$ von Pfosten A auf Pfosten B zu verschieben:

```
  Lege die oberste Scheibe von Pfosten A auf Pfosten B
  Lege die oberste Scheibe von Pfosten A auf Pfosten C
  Lege die oberste Scheibe von Pfosten B auf Pfosten C
  Lege die oberste Scheibe von Pfosten A auf Pfosten B
  Lege die oberste Scheibe von Pfosten C auf Pfosten A
  Lege die oberste Scheibe von Pfosten C auf Pfosten B
  Lege die oberste Scheibe von Pfosten A auf Pfosten B
```

Hinweis: Der Javadoc-Kommentar und die Signatur sehen wie folgt aus:

```
/**
 * Move a tower of given height from one pile (from) to another (to)
 *
 * @param height the height of the tower which shall be moved
 * @param from the pile on which the tower is located in the beginning
 * @param to  the pile on which the tower shall be in the end
 * @param help the third pile
 */
public void moveTower(int height, String from, String to, String help)
```