Übungen Kapitel 5

5.1 (*) Klasse Employee *(Klassenvariable, Klassenmethode)*

Schreiben Sie eine Klasse `Employee` mit folgenden Eigenschaften:

- einem Attribut `firstName`
- einem Attribut `lastName`
- einem sinnvollen Konstruktor
- getter/setter soweit sinnvoll (Lassen Sie sich die Methoden generieren!!)

Außerdem soll in dieser Klasse gezählt werden, wie viele Objekte dieser Klasse aktuell „leben“. Dazu erweitern Sie die Klasse um

- eine entsprechende Klassenvariable
- eine entsprechende Klassenmethode `getCount`(), die die Anzahl der erzeugten `Employee` Objekte zurückliefert
- sowie einer `toString` Methode, die sowohl die Instanzvariablen des Objekts als auch die Anzahl der erzeugten Objekte als String zurückliefert.

Testen Sie Ihre Klasse, in dem Sie sich z.B. 15 Objekte erzeugen lassen.

5.2 Fahrenheit *(static Methode)*

Schreiben Sie eine Klasse `Temperature` mit den beiden `static` float Methoden `toCelsius`, die für einen Fahrenheit Wert den äquivalenten Celsius Wert zurückliefert, sowie `toFahrenheit`, die für einen Celsius Wert, den entsprechenden Fahrenheitswert zurückliefert. Die Formeln lauten:

celsius = 5.0 / 9.0 * (fahrenheit – 32) bzw. fahrenheit = 9.0 / 5.0 * celsius + 32

Hier einige Beispielwerte:

```
>     Conversion Fahrenheit to Celsius:

      Fahrenheit: 10.0 Celsius: -12.222223
      Fahrenheit: 3.3 Celsius: -15.944446
      Fahrenheit: 5.5 Celsius: -14.722223
      Fahrenheit: 45.5 Celsius: 7.5000005

      Conversion Celsius to Fahrenheit:

      Celsius: -10.0 Fahrenheit: 14.0
      Celsius: 0.0 Fahrenheit: 32.0
      Celsius: 16.5 Fahrenheit: 61.699997
      Celsius: 36.0 Fahrenheit: 96.799995
```

5.3 () Triangle *(static Methode)*

Schreiben Sie eine Klasse `Triangle` mit einer `static` Methode `hypotenuse`, die für zwei Seiten a und b die Hypotenuse berechnet. Die Methode soll zwei Argumente vom Typ `double` haben und liefert den Wert als `double` zurück. Benutzen Sie die Wurzelfunktion `Math.sqrt`. Die Seiten Werte a und b werden durch den Bediener über `Scanner` eingegeben.

```
>Bitte a eingeben
23
Bitte b eingeben
34
Hypotenuse c= 41.048751503547585
```

5.4 (**) Zusatzaufgabe: Mathematische Hilfsfunktionen *(static Methode)*

Schreiben Sie eine Klasse `MathUtil` mit folgenden `static` Methoden:

a. `isEven`, die für eine Zahl angibt, ob Sie gerade oder ungerade ist.
b. `gcd`, die den größten gemeinsamen Teiler zwei ganzer Zahlen ermittelt (Algorithmus s.S 3.28)
c. `minimum3`, das für drei floating Point Zahlen, das Minimum berechnet. (Verwenden Sie hierzu `Math.min.`)
d. `integerPower,` die für eine Basis b und Exponent e b^e berechnet.
e. `printAsteriks`, die für einen Parameter n ein Quadrat aus * mit System.out.print ausgibt. Beispielsweise für n = 4 wird

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

ausgeben.

Für alle Methoden legen Sie den return Typ sowie die Parameter fest und testen die Methoden durch Eingabe von Zahlen durch `Console`.

Das arithmetische Mittel soll hierbei durch Aufruf der Methode `mean` berechnet werden.

5.5 (*) Primzahlen *(einfacher Algorithmus, static Methode)*

Primzahlen spielen in der Kryptographie eine wichtige Rolle. Insbesondere sind hier effiziente Programme notwendig, die Primzahlen erzeugen bzw. erkennen, ob eine Zahl Primzahl ist.

Entwickeln Sie einen Algorithmus, der für eine natürliche Zahl n >=2 feststellt, ob Sie eine Primzahl ist oder nicht. Prüfen Sie dazu alle möglichen Teiler t von n. Also der Algorithmus lautet also: n ist eine Primzahl, wenn es kein t mit 1 < t < n gibt, das n ohne Rest teilt (Modulo Operator %!!); andernfalls ist n keine Primzahl, sondern eine zusammengesetzte Zahl.

Implementieren Sie dazu eine Klasse `PrimTest` mit einer Methode `static boolean isPrime(int n)`. Das Programm gibt alle Primzahlen zwischen 2 und einer auf der Console gegebenen Obergrenze aus.

```
>Bitte Obergrenze eingeben
10
2
3
5
7
```

5.6 (**) Zusatzaufgabe: Primzahlen

Für die Kryptographie benötigt man große Primzahlen. Deshalb benötigt man Algorithmen, die große Primzahlen effizient erzeugen lassen. Eine wesentlich bessere Methode (als bei 5.4) ist das Ausprobieren von Zahlen (Pseudoprimzahlen), die mit hoher Wahrscheinlichkeit Primzahlen sind. Die Idee geht zurück auf den Satz von Fermat:

Für eine Primzahl n gilt: $a^{n-1} \bmod n = 1$ für alle $0 < a < n$

D.h. wir können für eine Zahl n zufallsmässig Basen a generieren und dann a und n dem Fermattest unterziehen.

Implementieren Sie dazu eine Klasse `Fermat` mit folgenden Methoden:

- **`static boolean fermatTest(int n, int a)`**
  Führt einen Fermat Test durch: Ist $a^{n-1} \bmod n = 1$?

  Verwenden Sie zum Potenzieren modulo n die `pow` Funktion in der Fermat Klasse (siehe Lösungshinweis). Für das willkürlich gewählte a = 17 ergibt sich:

```
System.out.println(fermatTest(999983, 17));     // true
System.out.println(fermatTest(999984, 17));     // false
```

Lösungshinweis: Im moodle liegt ein Code Skelett der Klasse `Fermat`. Insbesondere gibt es dort eine Funktion `pow` zum Potenzieren, die Sie für diese Methode direkt verwenden müssen.

- **`static boolean isPrime(int n)`**
  Führt mit (maximal) 100 zufällig gewählten Basen a den Fermat Test durch. Für die Zufallsgenerierung nehmen Sie wieder die Methode `Math.random()`. Achten Sie dabei, dass nur Werte für 0 < a < n erzeugt werden. Auch in diesem Fall wird für die beiden Zahlen 999983 sowie 999984 eine Basis gefunden:

```
System.out.println(isPrime(999983));     // true
System.out.println(isPrime(999984));     // false
```

- **`static int randomPrime()`**
  Generiert eine Pseudo-Primzahl n vom Typ int. Erzeugen Sie dazu so lange Zufallszahlen, bis eine den `isPrime` Test mit `true` übersteht.

```
>1155771943     // ist eine Primzahl
```

Lösungshinweis: Der größte mögliche Integerwert ist `Integer.MAX_VALUE`.

5.7 (*) Vertauschen *(Parameterübergabemechanismen)*

Schreiben Sie eine Klasse `CharacterUtil`:

a. Implementieren Sie eine Klassenmethode `vertausche` mit **zwei** Parametern `s` und `t` jeweils vom Typ `char[]` und **einem** Rückgabetyp `boolean`, der die characters von `s` und `t` vertauscht. Falls die Arrays nicht die gleiche Länge haben, wird abgebrochen und `false` zurückgeliefert, ansonsten `true`.

Die Anwendung dieser Methode zeigt folgendes Codebeispiel:

```
char[] s = { 'S', 'O', 'R', 'T', 'B' , 'Y'};
char[] t = { 'B', 'U', 'B', 'B', 'L' , 'E'};

CharacterUtil.vertausche(s,t);

// Jetzt hat s die Inhalte von t und t die Inhalte von
// s.
```

Lassen Sie sich jetzt die Inhalte von s und t ausgeben und prüfen die Korrektheit Ihrer Methode. Eine Beispielausgabe kann dann so aussehen:

```
>Vor Aufruf vertauschen:
 s=SORTBY
 t=BUBBLE

 Nach Aufruf vertauschen:
s=BUBBLE
 t=SORTBY
```

b. Testen Sie Ihre Klasse, wenn in `vertausche` **nur die Zeiger s und t** vertauscht werden. Erklären Sie das Resultat.

Bei allen folgenden Aufgaben dürfen Sie keine Schleifen verwenden!!!!

5.8 (*) Mystery *(Rekursion)*

Was tut die folgende Funktion? (Zeichnen Sie sich den callstack auf!!)

```
public class Mystery {

      public static int mystery(int a, int b) {
            if (b == 1)
                  return a;
            else
                  return a + mystery(a, b - 1);
      }

}
```

5.9 (*) Fakultät *(Rekursion)*

Implementieren Sie eine rekursive Funktion

- `static double fakultät(int n),`

die n! als Resultat zurückliefert. Lassen Sie sich dabei bei jedem Aufruf von Fakultät den aktuellen Parameter n ausgeben.

5.10 (*) Power *(Rekursion)*

Implementieren Sie eine rekursive Funktion

- `static double power (int base,int exponent),`

die base ^{exponent} als Resultat zurückliefert.

Nutzen Sie dabei aus, dass base ^{exponent} = base * base $^{exponent - 1}$!!

5.11 (*) Minimum *(Rekursion)*

Implementieren Sie eine rekursive Funktion

- `static int minimum (int[] array),`

die das Minimum der Elemente in `array` zurückliefert.

5.12 (*) Fibonacci *(Rekursion)*

Lösen Sie die Aufgabe 3.6 jetzt rekursiv.

5.13 (*) Zusatzaufgabe: gcd *(Rekursion)*

Implementieren Sie eine rekursive Funktion größter gemeinsamer Teiler

- `static int gcd (int a, int b),`

die den größten gemeinsamen Teiler zurückliefert.

5.14 (***) Zusatzaufgabe: ToBeDefined