Aufgaben zum Einstieg in das Programmierung mit Java

Diese Aufgaben sind für die Studierenden gedacht, die über wenige Vorerfahrungen in der Programmierung mit Java verfügen. In diesen Aufgaben wird zunächst der Umgang mit Schleifen eingeübt.

1 - Summe

Programmiere die Methode `int interval( int a, int b )`, die die Summe aller `int`-Werte in dem geschlossenen Intervall mit den Grenzen `a` und `b` berechnet und *zurückgibt*. Dabei bildet immer der kleinere der Werte aus `a` und `b` die untere Grenze, der größere Wert die oberere Grenze des Intervalls.

Beispiele:

`interval( 5, 11 )` gibt `56` zurück, da `5+6+7+8+9+10+11=56` gilt.
`interval( 11, 5 )` gibt ebenfalls `56` zurück, da `5+6+7+8+9+10+11=56` gilt.
`interval( 3, -1 )` gibt `5` zurück, da `-1+0+1+2+3=5` gilt.
`interval( 4, 4 )` gibt `4` zurück.
`interval( 0, 0 )` gibt `0` zurück.

2 - Quersumme

Schreibe die Methode `int digitSum( int x )`, die bei ihrer Ausführung eine ganze Zahl `x` als Argument erhält und die Quersumme von `x` *zurückgibt*. Die Quersumme ergibt sich als Summe der einzelnen Ziffern, die die Zahl bilden. Das Vorzeichen spielt bei der Berechnung der Quersumme keine Rolle.
Beispiele: Die Quersumme von `345` ist *3*`+4+5=12`, die Quersumme von `-17` ist `1+7=8`.
Hinweis: Im Dezimalsystem erhalten Sie die letzte Ziffer einer Zahl als Rest der Division durch `10`.

3 - Ziffern/Zahlen/Quersumme als Text

- Schreibe die Methode `String digitToString( int z1 )`, die bei ihrer Ausführung eine einstellige nicht-negative ganze Zahl `z1` – also eine einzelne Ziffer zwischen `0` und `9` – als Argument übergeben bekommt. Die Methode *gibt* einen `String` *zurück*, der den Namen der Ziffer enthält. Für andere Argumente soll der leere Text zurückgegeben werden.
  Beispiel: `digitToString( 7 )` soll als Ergebnis `"sieben"` liefern.
- Schreibe die Methode `String intToString( int z )`, die bei ihrer Ausführung eine ganze Zahl als Argument erhält und einen String *zurückgibt*. Der `String` enthält die Folge der Namen der Ziffern. Diese Namen werden durch `"-"` getrennt. Ist die übergebene Zahl negativ, wird das Wort `"minus"` vorangestellt. Benutze die Methode `digitToString`.
  Beispiel: Das Argument `-34` soll zu folgendem Text führen: `"minus-drei-vier"`.
- Schreibe nun die Methode `String digitSumToString( int x )`, die die Methoden `digitSum` und `intToString` nutzt. `digitSumToString` erhält bei ihrer Ausführung eine ganze Zahl `x` als Argument übergeben und *gibt* den Wert der Quersumme als Folge der Namen der Ziffern *zurück*.

4 - *Harshad*-Zahlen

- Eine natürliche Zahl heißt *Harshad*-Zahl, wenn Sie durch ihre Quersumme teilbar ist.
  Schreibe die Methode `boolean isHarshad( int z )`, die bei ihrer Ausführung eine ganze Zahl `z` als Argument erhält und `true` zurückgibt, falls `z` eine Harshad-Zahl ist und sonst `false`.
  Beispiel: `777` ist durch `7 + 7 + 7 = 21` teilbar und damit eine *Harshad*-Zahl.
- Schreibe die Methode `void computeHarshads( int n )`, die bei ihrer Ausführung eine ganze Zahl `n` als Argument erhält und alle *Harshad*-Zahlen von 1 bis bis `n` bestimmt und am Bildschirm *ausgibt*.

5 - *Eulersche* Zahl

Programmieren Sie eine Methode `double computeEuler( int n )`, die die Eulersche Zahl `e = 2,71...` berechnet. Die Berechnung soll nach der Addition des `n`-ten Summenglieds abbrechen und die berechnete Summe *zurückgeben*.

Die Eulersche Zahl e ergibt sich als: $e = \sum_{k=0}^{\infty} \frac{1}{k!} = \frac{1}{1} + \frac{1}{1} + \frac{1}{1 \cdot 2} + \frac{1}{1 \cdot 2 \cdot 3} + \frac{1}{1 \cdot 2 \cdot 3 \cdot 4} + \ldots$

Aufgaben zur Vorbereitung auf Testat 1

Als Vorbereitung auf das Testat 1 solltest Du unbedingt diese Aufgaben bearbeiten.

1 - Werte zählen

Schreibe die Methode `int countNegatives( int[] arr )`, die als Parameter ein Feld besitzt. Die Methode `countNegatives` zählt die negativen Werte in diesem Feld und *gibt* den ermittelten Wert *zurück*.

2 - Werte aufsummieren

Schreibe die Methode `int sumUpNegatives( int[] arr )`, die als Parameter ein Feld besitzt. Die Methode `sumUpNegatives` bildet die Summe der negativen Werte in diesem Feld und *gibt* den ermittelten Wert *zurück*.

3 - Bestimmen des Maximums in einem Feld

Schreibe die Methode `int maximum( int[] arr )`, die ein Feld als Parameter besitzt und die den größten Wert in diesem Feld bestimmt und *zurückgibt*.
Hinweis: Versuche, mit einem Durchlauf durch das Feld auszukommen.

4 - Bestimmen der Häufigkeit des Maximums in einem Feld

Schreibe die Methode `int countMaximum( int[] arr )`, die ein Feld von `int`-Werten als Parameter besitzt und die zählt, wie häufig der größte Wert vorkommt. Die ermittelte Anzahl wird *zurückgegeben*.

5 - Sortierung prüfen

Schreibe die Methode `boolean isSorted( int[] arr )`, die als Parameter ein Feld besitzt. Die Methode `isSorted` soll `true` *zurückgeben*, falls die im Feld enthaltenen Werte aufsteigend sortiert sind; sonst wird `false` *zurückgegeben*.

6 - Palindrome erkennen

Ein *Palindrom* ist eine Folge von `int`-Werten, die vorwärts und rückwärts gelesen die identische Ziffernfolge ergibt.

Beispiele: Die Folgen `12 34 78 34 12` oder `5 17 88 88 17 5` sind Palindrome.

Schreibe die Methode `boolean checkArray( int[] arr )`, die für das als Parameter übergebene Feld bestimmt, ob die Folge der Zahlen ein Palindrom bildet. Die Methode *gibt* einen Wert des Typs `boolean` *zurück*.

7 - Erhöhen der Inhalte eines Feldes

Schreibe die Methode `int[] increaseArray( int[] arr, int z )`, die als Parameter ein Feld und einen `int`-Wert `z` besitzt. Die Methode `increaseArray` erhöht alle Werte des Feldes um den Wert `z` und *gibt* das veränderte Feld *zurück*.

Beispiel: Werden ein Feld mit den Elementen `80,7,1,56,11,72,43,37` als erstes und der Wert `17` als zweites Argument übergeben, so wird dieses Feld verändert und als `97,24,18,73,28,89,60,54` *zurückgegeben*.

8 - Bedingtes Verdoppeln der Inhalte eines Feldes

Schreibe die Methode `int[] doubleIfContainsPositive( int[] arr )`, die als Parameter ein Feld des Typs `int` besitzt. Die Methode `doubleIfContainsPositive` verdoppelt alle Werte des Feldes, falls in dem Feld *mindestens ein* positiver Wert vorkommt; sonst bleibt das Feld unverändert. Das (veränderte) Feld soll *zurückgegeben* werden.

Beispiel: Wird ein Feld mit den Elementen `0,-7,1,5,-1,2,4` als Argument übergeben, so wird dieses Feld verändet und als `0,-14,2,10,-2,4,8` *zurückgegeben*.

9 - Erzeugen eines Textes

Schreibe die Methode `String toString( int[] arr )`, die als Parameter ein Feld besitzt. Die Methode `toString` erzeugt einen Text, der alle Inhalte des Feldes in der Reihenfolge ihres Auftretens durch Kommas getrennt enthält. Der erzeugte Text wird *zurückgegeben*.

10 - Erzeugen eines Feldes mit drei Elementen

Schreibe die Methode `int[] copyStartingValues( int[] arr )`, die als Parameter ein Feld besitzt. Die Methode `copyStartingValues` erzeugt ein neues Feld, das in seinen Elementen genau die ersten drei Werte des als Argument übergebenen Feldes enthält, falls dieses mehr als zwei Elemente besitzt. Das erzeugte Feld wird *zurückgegeben*. Besitzt das als Argument übergebene Feld weniger als drei Elemente, wird es vollständig kopiert.

Beispiel: Wird ein Feld mit den Elementen `80,7,1,56,11,72,43,37` als Argument übergeben, so wird ein neues Feld mit den Werten `80,7,1` *zurückgegeben*.

11 - Erzeugen eines Feldes mit ausgesuchten Inhalten

Schreibe die Methode `int[] selectNegatives( int[] arr )`, die als Parameter ein Feld besitzt. Die Methode `selectNegatives` *gibt* ein Feld *zurück*, in dem ausschließlich die negativen Werte des als Argument übergebenen Feldes enthalten sind. Die Methode `countNegatives` kann dazu benutzt werden, die Größe des zurückgegebenen Feldes zu bestimmen.

Beispiel: Wird ein Feld mit den Elementen `80,-7,1,56,-11,-72,0,37` als Argument übergeben, so wird ein neues Feld mit den Werten `-7,-11,-72` *zurückgegeben*.

12 - Erzeugen eines Feldes mit invertiertem Inhalt

Schreibe die Methode `int[] copyAndInvert( int[] arr )`, die als Parameter ein Feld besitzt. Die Methode `copyAndInvert` *gibt* ein Feld *zurück*, das die Werte des als Argument übergebenen Feldes in umgekehrter Reihenfolge enthält.

Beispiel: Wird ein Feld mit den Elementen `80,-7,1,56,-11,-72,0,37` als Argument übergeben, so wird ein neues Feld mit

den Werten `37,0,-72,-11,56,1,-7,80` *zurückgegeben*.

13 - Zusammenführen von Feldern

Schreibe die Methode `int[] addArrays( int[] arr1, int[] arr2 )`, die zwei `int`-Felder als Parameter besitzt. Haben diese Felder die gleiche Länge, so werden die Werte am gleichen Index aus beiden Feldern addiert und die Summe unter diesem Index in einem dritten `int`-Feld abgelegt werden. Dieses Feld wird als Ergebnis des Aufrufs von `addArrays` *zurückgegeben*. Unterscheiden sich die als Argumente übergebene Felder in ihrer Länge, wird ein leeres Feld *zurückgegeben*.

14 - Zählen von Folgen

Schreibe die Methode `int countSequences( int[] arr )`, die ein Feld als Parameter besitzt. Die Methode `countSequences` ermittelt die Anzahl der im Feld enthaltenen Zahlenfolgen, in denen **nicht** der Wert `0` vorkommt. Eine solche Zahlenfolge endet immer mit dem Auftreten einer `0` oder dem Ende des Feldes. Die ermittelte Anzahl wird von der Methode *zurückgegeben*.

Beispiel: Wird ein Feld mit den Elementen `80,7,1,0,11,72,0,0,37,61` als Argument übergeben, so wird der Aufruf von `countSequences` als Ergebnis `3` liefern, da die Folgen `80,7,1` und `11,72` und `37,61` auftreten.

15 - Analyse eines Felds

Schreibe die Methode `boolean twoTimes( int[] arr )`, die `true` *zurückgibt*, wenn das Feld `arr` mindestens zwei Elemente besitzt und jeder im Feld vorkommende Wert *genau* zweimal auftritt. Sonst wird `false` *zurückgegeben*.

Beispiele:	*Feld*	*Ergebnis*
	`2 3 4 3 7 7 4 1 2 1`	`true`
	`2 2 8 8 5 5 3 3 9 9`	`true`
	`2 3 2 3 5 5 1 2 1 2`	`false`, da der Wert `2` zu häufig vorkommt
	`2 3 3 5 5 4 1 2 1`	`false`, da der Wert `4` nur einmal vorkommt