Aufgaben zur Übung Software Engineering SE-1 A4

Wiederholung Programmierung mit Java, Punkte: 10

Erstellen Sie ein neues Java-Projekt *playground* und darin ein Paket *application*. Sie finden in Moodle eine Datei *PlayNumbers.java*, die Sie bitte in dieses Paket kopieren. Führen Sie die folgenden Aufgaben in dieser Datei aus.

Gegeben ist ein Array mit Integer Zahlen:

```
final int[] numbers = {-2, 4, 9, 4, -3, 4, 9, 5};
```

1.) Implementieren Sie eine Methode, welche die Zahlen in `numbers` addiert:

```
int sum(int[] numbers) { ... }
```

Bei Ausführung des Programms erscheint das Ergebnis: [1 Pkt]

```
int result = sum(numbers);
System.out.println(String.format(" - sum(numbers) -> %d", result));
//
Ausgabe ==> sum(numbers) -> 30      // Ausgabe
```

Im Weiteren werden Methoden über Test-Methoden aufgerufen, welche Ausgaben zusammenfassen:

```
test0("sum", () -> sum(numbers));    // Aufruf von sum() über test-Methode
//
Ausgabe ==> sum(numbers) -> 30      // Ausgabe
```

2.) Implementieren Sie eine Methode, die nur positive, gerade Zahlen in `numbers` addiert:

```
int sumPositiveEvenNumbers(int[] numbers) { ... }
```

[1 Pkt]

```
test0("sumPositiveEvenNumbers", () -> sumPositiveEvenNumbers (numbers));
//
Ausgabe ==> sumPositiveEvenNumbers(numbers) -> 12  // Ausgabe
```

3.) Implementieren Sie weitere Methoden:

```
int findFirst(int x, int[] numbers) { ... }
```

welche den *Index* des ersten Auftretens von `x` in `numbers` liefert oder -1, wenn `x` nicht in `numbers` ist.

```
int findLast(int x, int[] numbers) { ... }
```

welche den Index des letzten Auftretens von `x` in `numbers` liefert oder -1, wenn `x` nicht in `numbers` ist.

```
int[] findAll(int x, int[] numbers) { ... }
```

welche die Indizes aller `x` in `numbers` liefert oder ein leeres Array [ ] (nicht *null*), wenn `x` nicht in `numbers` ist.

Ergebnisse bzw. Ausgaben für Beispiel-Zahlen `x = {4, -3, 1}` sind:

```
testing: x = 4
 - findFirst(4, numbers) -> 1
 - findLast(4, numbers) -> 5
 - findAll(4, numbers) -> [1, 3, 5]

testing: x = -3
 - findFirst(-3, numbers) -> 4
 - findLast(-3, numbers) -> 4
 - findAll(-3, numbers) -> [4]

testing: x = 1  // nicht gefunden
 - findFirst(1, numbers) -> -1
 - findLast(1, numbers) -> -1
 - findAll(1, numbers) -> []
```

[3 Pkt]

4.) Implementieren Sie eine Methode: [1 Pkt]

```
int[][] findAllAdjacent(int x, int y, int[] numbers)
            { ... }
```

welche benachbarte Zahlen `x` und `y` in `numbers` findet und als Ergebnis Tupel der Indizes **[i_x, i_y]** liefert.

Tupel können als kurze 2- oder *n*-stellige `int`-Arrays realisiert werden. Da es mehrere benachbarte Zahlenpaare geben kann, liefert die Methode ein Array von Tupeln (`int[][]`).

Ausgaben für Beispiel-Werte (`x, y`) mit: `(4,9), (9,4), (2,3)` sind:

```
- findAllAdjacent(4, 9, numbers)
  -> [[1, 2], [5, 6]]
- findAllAdjacent(9, 4, numbers)
  -> [[2, 3]]
- findAllAdjacent(2, 3, numbers)
  -> []   // leer, kein Match
```

5.) Implementieren Sie eine Methode, die feststellt, ob sich eine gesuchte Zahl `sum` als Summe zweier Zahlen aus `numbers` bilden lässt (jede Zahl in `numbers` kann nur einmal verwendet werden).

```
int[][] findSums(int sum, int[] numbers) { ... }
```

Die Methode liefert ein Array von Tupeln mit allen Paaren `x` und `y` in `numbers`, die als Ergebnis `sum` ergeben.

Tupel enthalten hier die Zahlen, nicht die Indizes.

Verwenden Sie das zweite Array für `numbers`:

```
final int[] numbers_2 = {8, 10, 2, 14, 4};
```

für den Test. [1 Pkt]

Ausgaben für Beispiel-Werte:

```
- findSums( 10, [8, 10, 2, 14, 4])
  -> [8, 2]     // 10 = 8 + 2

- findSums( 12, [8, 10, 2, 14, 4])
  -> [8, 4]     // 12 = 8 + 4
  -> [10, 2]    // 12 = 10 + 2

- findSums( 15, [8, 10, 2, 14, 4])
  -> []   // leer, kein Match
```

Die Ausgabe kann direkt über das zurückgegebene `int[][]` erfolgen oder über die Test-Methode *test4()*.

```
int[][] results = findSums( 10, numbers_2);     // neue Zahlenliste numbers_2
System.out.println(String.format("\nfindSums(%d, %s)", sum, Arrays.toString(numbers_2)));
for(int i=0; i < results.length; i++) {
       System.out.println(String.format(" - %s", Arrays.toString(results[i])));
}
```

6.) Implementieren Sie eine Methode, die alle Kombinationen feststellt, aus denen sich eine Zahl `sum` als Summe von Zahlen aus `numbers` bilden lässt (jede Zahl kann nur einmal verwendet werden).

```
int[][] findAllSums(int sum, int[] numbers) ...
```

Die Methode soll *n*-stellige Tupel mit Zahlen aus `numbers` liefern, deren Summe die gesuchte Zahl `sum` ergibt.

[2 Pkt]

Ausgaben für Beispiel-Werte:

```
- findAllSums( 14, [8, 10, 2, 14, 4])
  -> [[14], [10, 4], [8, 2, 4]]

- findAllSums( 20, [8, 10, 2, 14, 4])
  -> [[8, 10, 2], [2, 14, 4]]

- findAllSums( 32, [8, 10, 2, 14, 4])
  -> [[8, 10, 14]]
```

Beantworten Sie die folgenden Fragen:

a) Welches Grundprinzip unterliegt der Lösung (Teilmenge, Potenzmenge, Permutation)?

b) Wieviel Kombinationen gibt es, wenn die Ausgangszahlenliste *n* Elemente hat?

c) Wie kann man die Kombinationen erzeugen? [1 Pkt]