Java Intro Workshop

4 Aufgaben zu Klassen

Aufgabe 1: SuperHero

In dieser Aufgabe soll eine Klasse `SuperHero` implementiert werden, die Superhelden mit ihren Namen, ihren Superkräften und ihrer Stärke modelliert. Zudem soll eine Methode `boolean isStrongerThan(SuperHero)` prüfen, ob ein Superheld stärker als ein anderer ist. Schließlich ist eine allgemeingültige Prüfung zu realisieren, die den stärksten Superhelden aus einer Menge zurückgibt. Dazu soll eine Methode `SuperHero strongestOf(SuperHero...)` mithilfe von Var Args, einer variablen Parameterliste, erstellt werden. Folgendes Hauptprogramm zeigt die Verwendung und mögliche Aufrufe – variieren Sie gern die Eigenschaften.

```
public static void main(String[] args)
{
    SuperHero superMan =
              new SuperHero("Superman", "Kryptonite-Power", 1_000);
    SuperHero batMan = new SuperHero("Batman", "Techno-Power", 100);
    SuperHero ironMan = new SuperHero("Ironman", "Techno-Power", 500);
    System.out.println("Superman stronger than Batman? " +
                       superMan.isStrongerThan(batMan));
    System.out.println(SuperHero.strongestOf(superMan, batMan, ironMan));
}
```

Aufgabe 2: Counter

Nachdem die Grundbegriffe beim objektorientierten Entwurf mit Java bekannt sind, soll nun ein Zähler als Klasse entworfen werden, der folgende Anforderungen erfüllt:

1. Er lässt sich auf den Wert 0 zurücksetzen.
2. Er lässt sich um eins erhöhen.
3. Der aktuelle Wert lässt sich abfragen.

Dabei existiert bereits das folgende, rudimentäre Grundgerüst:

```
public class Counter
{
    public int count = 0;

    public Counter()
    {
    }
```

```
    public void setCounter(int count)
    {
        count = count;
    }
}
```

Das Attribut `count` speichert den Zähler und kann von überall abgefragt und verändert werden. Das Rücksetzen erfolgt durch Übergabe von 0. Die obige Implementierung sollen Sie nun korrigieren, erweitern und verbessern, indem Sie zu den Anforderungen passende Methoden implementieren und als Kür das Attribut „verstecken“.

Aufgabe 3: Vererbung

Erstellen Sie zwei Basisklassen `ExcludeFilter` und `IncludeFilter` sowie basierend darauf einen SPAM-Filter und einen Namensfilter, die jeweils eine `filter()`-Methode anbieten und wie folgt aufgerufen werden:

```
public static void main(String[] args)
{
    IFilter spamfilter = new SPAMFilter();
    System.out.println(spamfilter.filter(List.of("SPAM",
"DAS", "IST", "SPAM", "SPAM", "DETECTED")));

    IFilter namefilter = new NameFilter();
    System.out.println(namefilter.filter(List.of("DAS",
"SIND", "MICHAEL", "UND", "SOPHIE")));
}
```

Das gewünschte Ergebnis ist wie folgt:

```
[DAS, IST, DETECTED]
[MICHAEL, SOPHIE]
```

BONUS: Führen Sie nun noch eine abstrakte Basisklasse `AbstractFilter` oder ein Interface `IFilter` ein, die lediglich die Signatur der Methode vorgibt. Modifizieren Sie die Konstruktion und Aufrufe passend.

5 Aufgaben zu Collections

Aufgabe 1: Tennisverein-Mitgliederliste

Nehmen wir an, wir verwalten die Mitglieder des Tennisvereins in Form einer Liste. Implementieren Sie folgenden Registrierungsprozess: Zunächst melden sich Michael, Tim und Werner an. Prüfen Sie, ob sich Jana schon angemeldet hat. Nun registriert sich Andreas. Schließlich melden sich Lili, Jana und Natalija an. Geben Sie die Anzahl der Mitglieder aus.

Aufgabe 2: Befreundete Zahlen

Zwei Zahlen n1 und n2 heißen befreundet, wenn die Summe ihrer Teiler der jeweils anderen Zahl entspricht:

sum(divisors(n1)) = n2

sum(divisors(n2)) = n1

Schreiben Sie eine Methode `boolean areFriends(int value1, int value2)` zur Berechnung, ob die zwei Zahlen befreundeten Zahlen sind.

Aufgabe 3: Check Magic Triangle

Schreiben Sie eine Methode `boolean isMagicTriangle(List<Integer> values)`, die prüft, ob eine Folge von Zahlen ein magisches Dreieck bildet. Ein solches ist definiert als ein Dreieck, bei dem die jeweiligen Summen der Werte der drei Seiten alle gleich sein müssen. Beginnen Sie mit einer Vereinfachung für die Seitenlänge 3 und einer Methode `boolean isMagic6(List<Integer> values)`.

Beispiele Nachfolgend ist das für je ein Dreieck der Seitenlänge drei und der Seitenlänge vier gezeigt:

```
  1          2
 6 5        8 5
2 4 3      4   9
          3 7 6 1
```

Damit ergeben sich folgende Seiten und Summen:

Eingabe	Werte 1	Werte 2
Seite 1	1 + 5 + 3 = 9	2 + 5 + 9 + 1 = 17
Seite 2	3 + 4 + 2 = 9	1 + 6 + 7 + 3 = 17
Seite 3	2 + 6 + 1 = 9	3 + 4 + 8 + 2 = 17

Aufgabe 4: Mengen und ihre Operationen

Zunächst ist es Ihre Aufgabe, Duplikate aus einer Liste mit Städtenamen zu entfernen:

```
List<String> cities = list.of("Kiel", "Hamburg", "Zürich", "Bremen",
                              "Hamburg", "Zürich", "Kiel", "Bremen");
```

Nun sollen Sie für folgende Zahlen die Mengenoperationen berechnen:

```
{ 1, 1, 2, 3, 5, 8, 13, 21 }
{ 1, 2, 3, 4, 5, 6, 7}
```

Aufgabe 5: Doppelte Buchstaben entfernen

Schreiben Sie eine Funktion `String remove_duplicates(String input)`, die in einem gegebenen Text jeden Buchstaben nur einmal behält, also alle späteren doppelten unabhängig von Groß- und Kleinschreibung löscht. Dabei soll aber die ursprüngliche Reihenfolge der Buchstaben beibehalten werden.

Eingabe	Resultat
"bananas"	"bans"
"lalalamama"	"lam"
"MICHAEL	"MICHAEL"

Aufgabe 6: Duplikate entfernen

Aus einer Liste sollen Sie die doppelten Einträge entfernen. Dabei gilt die Randbedingung, dass die ursprüngliche Reihenfolge bestehen bleibt. Schreiben Sie dazu eine Funktion `List<Integer> removeDuplicates(List<Integer> values)`.

Eingabe	Resultat
[1, 1, 2, 3, 4, 1, 2, 3]	[1, 2, 3, 4]
[7, 5, 3, 5, 1]	[7, 5, 3, 1]
[1, 1, 1, 1]	[1]

Tipp: Kombinieren Sie Listen und Mengen geeignet.

Aufgabe 7: Zahl als Text

Schreiben Sie eine Methode `String numberAsText(int n)`, die für eine gegebene positive Zahl vom Typ `int` die jeweiligen Ziffern in korrespondierenden Text umwandelt.

```
System.out.println(numberAsText(721971));
```

Das sollte folgende Ausgabe produzieren:

```
SEVEN TWO ONE NINE SEVEN ONE
```

Aufgabe 8: Morse Code

Schreiben Sie eine Methode `String toMorseCode(String input)`, die einen übergebenen Text in Morsezeichen übersetzen kann. Diese bestehen aus Sequenzen von ein bis vier kurzen und langen Tönen pro Buchstabe, symbolisiert durch ' .' oder ' -'. Zur leichteren Unterscheidbarkeit soll zwischen jedem Ton ein Leerzeichen und zwischen jeder Buchstabentonfolge jeweils drei Leerzeichen Abstand sein – ansonsten würden sich S (...) und EEE (...) nicht voneinander unterscheiden lassen.

Beschränken Sie sich der Einfachheit halber auf die Buchstaben E, O, S, T, W mit folgender Codierung

Buchstabe	Morsecode
E	.
O	- - -
S	. . .
T	-
W	. - -

Hier ein paar Beispiele für mögliche Umwandlungen:

Eingabe	Resultat
SOS	. . . - - - . . .
TWEET	- . - - . . -
WEST	. - - . . . . -

Aufgabe 9: Häufigkeiten von Namen

Stellen Sie sich vor, es wäre eine Liste mit Namen gegeben:

```
jshell> var names = List.of("Tim", "Tom", "Mike", "Jim", "Tim",
                            "Mike", "James", "Mike")
names ==> [Tim, Tom, Mike, Jim, Tim, Mike, James, Mike]
```

Nun wollen Sie wissen, welcher Name am häufigsten vorkommt bzw. genauer, für alle Namen deren Anzahl ermitteln. Bereiten Sie das Ergebnis als Map etwa wie folgt auf:

```
{'Tim': 2, 'Tom': 1, 'Mike': 3, 'Jim': 1, 'James': 1}
```

Aufgabe 10: Anagramm

Als Anagramm bezeichnet man zwei Strings, die dieselben Buchstaben in der gleichen Häufigkeit enthalten. Dabei soll Groß- und Kleinschreibung keinen Unterschied machen. Schreiben Sie eine Funktion `boolean isAnagram(String str1, String str2)`.

Eingabe 1	Eingabe 2	Resultat
"Otto"	"Toto"	True
"Mary"	"Army"	True
"Ananas"	"Bananas"	False