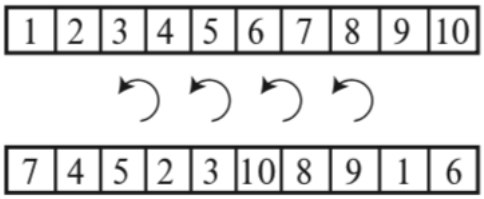
Java Intro Workshop

3 Aufgaben Arrays

Aufgabe 1: Durcheinanderwürfeln eines Arrays

Bei dieser Aufgabe geht es darum, die Elemente eines bestehenden Arrays durcheinanderzuwürfeln. Das ist nützlich, wenn man eine zufällige Verteilung benötigt. Schreiben Sie dazu eine Methode `void shuffle(int[] values)`.

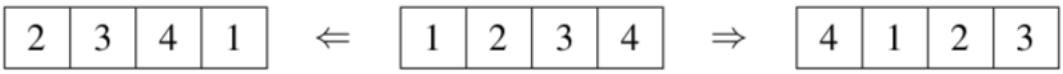



```
public static void main(String[] args)
{
    int[] sorted = { 1, 2, 3, 4, 5, 6, 7};
    shuffle(sorted);
    System.out.println(Arrays.toString(sorted));

    int[] primes = { 2, 3, 5, 7, 11, 13};
    shuffle(primes);
    System.out.println(Arrays.toString(primes));
}
```

Aufgabe 2: Rotation um eine oder mehrere Positionen

In dieser Aufgabe besteht die Problemstellung im Rotieren eines Arrays um n Positionen nach links bzw. rechts. Dabei sollen die Elemente zyklisch am Anfang bzw. Ende nachgeschoben werden. Das gewünschte Vorgehen ist nachfolgend für eine Rotation um eine Positionen visualisiert, wobei das mittlere Array die Ausgangsbasis bildet:



Aufgabe 3: Arrays kombinieren

Es seien zwei Arrays gegeben. Die Aufgabe besteht nun darin, ein neues Array mit allen Werten zu erstellen. Dabei sollen zunächst alle Elemente aus dem zweiten Array und dann alle aus dem ersten in das Ergebnis übernommen werden. Nachfolgend ist dies visualisiert:

```
public static void main(String[] args)
{
    String[] first = { "neue", "Array", "Funktion" };
    String[] second = { "Dies", "ist", "eine" };

    String[] result = arraySpecialConcat(first, second);
    System.out.println(Arrays.toString(result));
}
```

Liefert folgendes Ergebnis

```
[Dies, ist, eine, neue, Array, Funktion
```

Aufgabe 4: Array-Werte bilden ein Palindrom

Schreiben Sie eine Methode `boolean isPalindrome(int[] values)`, die prüft, ob die Werte ein Palindrom bilden.

Aufgabe 5: Arrays durchsuchen

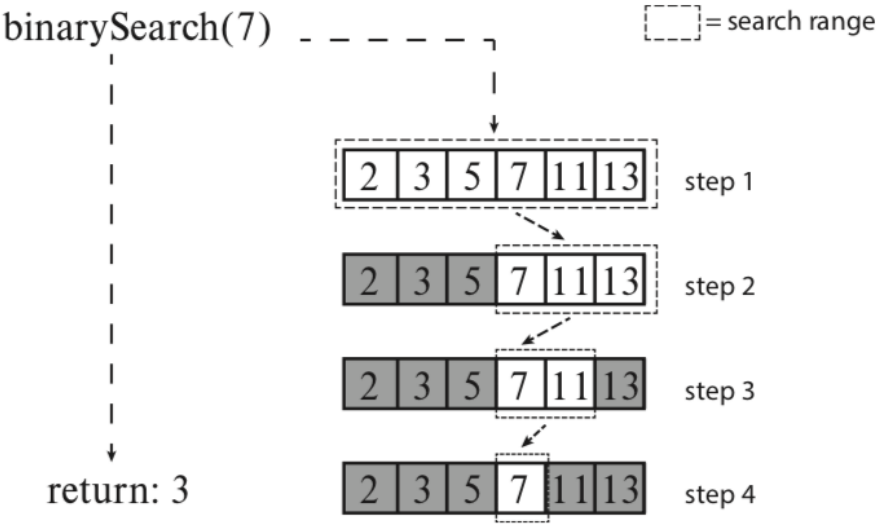
Es sollen zwei Methoden `int indexOf(int[] values, int searchFor)` und `int lastIndexOf(int[] values, int searchFor)` zur Suche nach Werten vom Anfang und vom Ende her erstellt werden.

```
public static void main(String[] args)
{
    int[] values = { 1, 2, 3, 4, 7, 2, 3, 2 };

    System.out.println(indexOf(values, 2));
    System.out.println(lastIndexOf(values, 2));
}
```

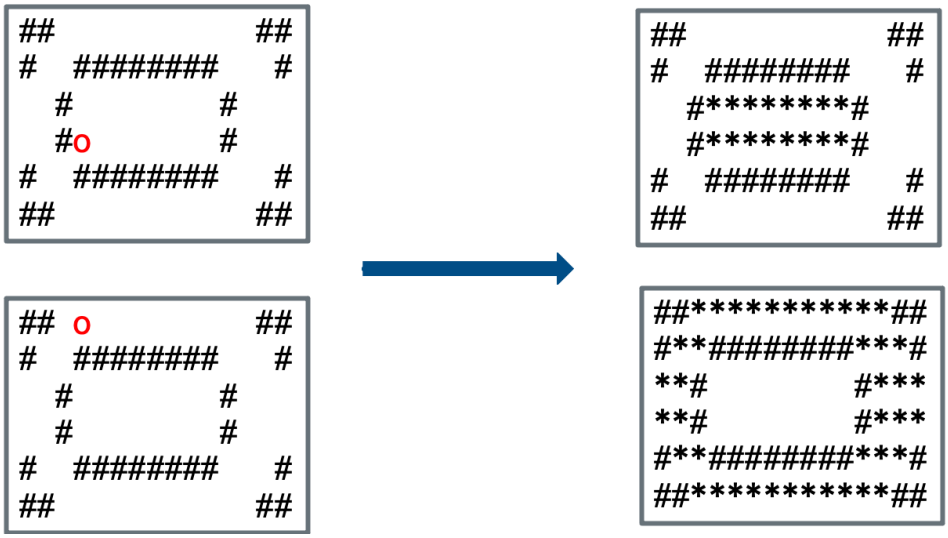
Aufgabe 6: Binärsuche mit Rekursion

Schreiben Sie eine Methode `int binarySearch(int[] values, int desiredValue)`, die in einem sortierten Array nach einem gewünschten Wert sucht und bei nicht Existenz den Wert -1 zurückgibt. Es soll die Binärsuche genutzt werden, die das Array bei jedem Schritt in zwei Hälften teilt und in dem korrekten Teilbereich weitersucht.



Aufgabe 7: Flood-Fill

Schreiben Sie eine Methode `void floodFill(char[][] values2dim, int startX, int startY)`, die in einem verschachtelten Array alle freien Felder mit einem bestimmten Wert befüllt. Nachfolgend ist der Füllvorgang für das Zeichen '*' gezeigt. Das Füllen beginnt an einer vorgegebenen Position, etwa in der linken oberen Ecke, und wird dann so lange in alle vier Himmelsrichtungen fortgesetzt, bis die Grenzen der Listen oder eine Begrenzung in Form eines anderen Zeichens gefunden wird:



Aufgabe 8 – Die Klasse `Arrays`

Ermitteln Sie, ob der durch `search` beschriebene Text im Originaltext `originaltext` vorkommt. Wandeln Sie dazu die Strings zunächst in den Typ `byte[]` um und finden Sie die Position der Übereinstimmung von `search`:

```
final String originaltext = "BLABLASECRET-INFO:42BLABLA";
final String search = "SECRET-INFO:42";
```

Aufgabe 9 – Die Klasse `Arrays`

Ermitteln Sie für die beiden wie folgt gegebenen Arrays

```
final byte[] first =  { 1,1,0,1,1,0,1,1,1,1,0,1,1 };
final byte[] second = { 1,1,0,1,1,0,1,0,1,1,1,1,1 };
```

a) die erste Abweichung und
b) die darauffolgende Abweichung.

Aufgabe 10 – Die Klasse `Arrays`

Vergleichen Sie die beiden wie folgt gegebenen Arrays:

```
final byte[] first =  "ABCDEFGHIJk".getBytes();
final byte[] second = "XYZABCDEXYZ".getBytes();
```

a) Welches ist «grösser»?
b) Ab welcher Position ist `first` grösser als `second`, wenn man von `second` immer bei jedem weiteren Vergleich einen Buchstaben vorne entfernt. Protokolliere zum besseren Verständnis die verglichenen Werte für alle Start-Positionen.

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.