

Tutorium Hinterreiter/Stein/Gallinger

Aufgabe 16a:

```
// Necessary packages/classes:
import java.time.LocalDate;
import java.time.Period;
// https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/time/package-summary.html
//

public class Aufgabe_16a {

    // -----
    public static void main(String[] args) {

        // 1. Erstellen Sie ein LocalDate-Objekt für das aktuelle Datum
        LocalDate heute = LocalDate.now();

        // 2. Erstellen Sie ein LocalDate-Objekt für den vorläufigen Termin der ersten Klausur
        // 1. August 2023
        LocalDate klausurtermin = LocalDate.of(2023, 8, 1);

        // 3. Erstellen Sie ein Period-Objekt, das die Zeitspanne vom aktuellen Datum zum Prüfungstermin enthält
        Period bisZurKlausur = Period.between(heute, klausurtermin);

        // 4. Die verbleibenden Monate und Tage auf der Kommandozeile ausgeben
        System.out.println("Verbleibende Monate: " + bisZurKlausur.getMonths());
        System.out.println("Verbleibende Tage: " + bisZurKlausur.getDays());
    }

    // -----
}
```

Aufgabe 16b:

```
// Necessary packages/classes:
import java.time.LocalDate;
import java.time.DayOfWeek;
// https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/time/package-summary.html
//

public class Aufgabe_16b {

    // -----

    public static void main(String[] args) {

        // aktuelles Datum holen
        LocalDate heute = LocalDate.now();

        // Anzahl der Tage bis zum Monatsende bestimmen
        LocalDate monatsende = heute.withDayOfMonth(heute.lengthOfMonth());
        int tageBisMonatsende = monatsende.getDayOfMonth() - heute.getDayOfMonth();

        // Ausgabe der Anzahl der Tage bis zum Monatsende
        System.out.println("Tage bis Monatsende: " + tageBisMonatsende);

        // Samstage und Sonntage bis Monatsende bestimmen
        int samstage = 0, sonntage = 0;

        for (LocalDate date = heute; date.isBefore(monatsende.plusDays(1)); date = date.plusDays(1)) {
            if (date.getDayOfWeek() == DayOfWeek.SATURDAY) {
                samstage = samstage + 1;
            }
        }
    }
}
```

oh gott bitte
while, wenn kein int
Iterator da ist

```

        else if (date.getDayOfWeek() == DayOfWeek.SUNDAY) {
            sonntage = sonntage + 1;
        }
    }

    // Ausgabe der Anzahl der Samstage und Sonntage
    System.out.println("Verbleibende Samstage: " + samstage);
    System.out.println("Verbleibende Sonntage: " + sonntage);
}

// -----
}

```



Aufgabe 16c:

```

// Necessary packages/classes:
import java.time.LocalDateTime;
import java.time.Duration;
import java.util.concurrent.ThreadLocalRandom;
// https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/time/package-summary.html
//

public class Aufgabe_16c {

    // -----

    // Methode zur Generierung einer zufälligen Uhrzeit
    public static LocalDateTime zufaelligeUhrzeit() {
        int stunde = ThreadLocalRandom.current().nextInt(0, 24);
        int minute = ThreadLocalRandom.current().nextInt(0, 60);

        // Return
        return LocalDateTime.of(stunde, minute);
    }

    // -----

    public static void main(String[] args) {

        // Zwei zufällige Zeiten bestimmen
        LocalDateTime t1 = zufaelligeUhrzeit();
        LocalDateTime t2 = zufaelligeUhrzeit();

        // Speichern Sie die frühere der beiden Zeiten in der Variable t1, die spätere in der Variable t2
        if (t2.isBefore(t1)) {
            LocalDateTime temp = t1;
            t1 = t2;
            t2 = temp;
        }

        // Geben Sie beide Zeiten aus
        System.out.println("Zeit t1: " + t1);
        System.out.println("Zeit t2: " + t2);

        // Bestimmen Sie dann, solange zwischen t1 und t2 mehr als 15 Minuten liegen, eine neue zufällige Zeit
        while (Duration.between(t1, t2).toMinutes() > 15) {
            LocalDateTime neueZeit = zufaelligeUhrzeit();

            // mit der sie t2 überschreiben, sofern die neue Zeit zwischen t1 und dem alten Wert von t2 liegt
            if (neueZeit.isAfter(t1) && neueZeit.isBefore(t2)) {
                t2 = neueZeit;
            }
        }

        // Geben Sie am Ende die neue Zeiten jeweils in einer eigenen Zeile auf der Kommandozeile aus
        System.out.println("Neue Zeit t1: " + t1);
    }
}

```

```

        System.out.println("Neue Zeit t2: " + t2);
    }

    // -----
}

```



Aufgabe 17 a & b:

```

// Necessary packages/classes:
import java.util.Random;
import java.util.ArrayList;
import java.util.Iterator;
// https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/util/package-summary.html
//

```

```

public class Aufgabe_17 {

```

```

    // -----

```

```

    // 17a. Schreiben Sie eine Klassenmethode randomString, die eine nat'urliche Zahl n erwartet
    // und ein String-Objekt der L'ange n zur'ueckgibt, der aus zuf'alligen lateinischen Gro'ß- und
    // Kleinbuchstaben besteht.
    public static String randomString(int n) {

```

```

        // String initialisieren und random seed generieren
        StringBuilder sb = new StringBuilder(n); // StringBuilder: Sequence of characters
        Random rand = new Random(); // Random seed

```

```

        // Loop through um sb mit Buchstaben zu befu'llen
        for (int i = 0; i < n; i = i + 1) {

```

```

            // zuf'alligen Gro'ß- oder Kleinbuchstaben w'ahlen
            char c = (rand.nextBoolean()) ? // Generates either true or false
                (char) (rand.nextInt(26) + 'A') : // If true;
                (char) (rand.nextInt(26) + 'a'); // If false
            // (char) ist notwendig, da (rand...) ein int returned, StringBuilder braucht aber einen char

```

hier kann man auch einfach eine if-Anweisung nutzen

```

            // Buchstaben an sb anha'ngen
            sb.append(c);
        }

```

```

        // Return
        return sb.toString();
    }

```



```

    // -----

```

```

    public static void main(String[] args) {

```

```

        // 17b. Erzeugen Sie ein streng typisiertes ArrayList-Objekt f'ur String-Objekte und eine Variable
        // myList, die das Objekt referenziert
        ArrayList<String> myList = new ArrayList<>();

```

```

        // 1. Befu'llen Sie myList mit 1000 zuf'alligen Zeichenketten mit L'angen zwischen 3 und 7
        Random rand = new Random(); // random seed

```

```

        for (int i = 0; i < 1000; i = i + 1) {

```

```

            // zuf'allige L'ange zwischen 3 und 7 bestimmen
            int length = rand.nextInt(5) + 3;

```

```

            // Appends string to end of ArrayList
            myList.add(Aufgabe_17.randomString(length));
        }

```

besser:
main nach oben

```

// 2. Überprüfen Sie und geben Sie aus, ob myList die Zeichenkette "ABC" enthält.
System.out.println("Enthält 'ABC': " + myList.contains("ABC")); // contains returns true if element is in list

// 3. Erzeugen Sie ein Iterator-Objekt für myList und durchlaufen Sie alle Zeichenketten.
// Summieren Sie dabei die Länge der Zeichenketten auf und geben diese Gesamtlänge aus.

Iterator<String> iterator = myList.iterator();
// ("Performs the given action for each remaining element until all elements have been processed or the action
throws an exception.")
// https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/util/Iterator.html
int gesamtlänge = 0;

// while loop - Würde sich auch über for loop lösen lassen, aber while einfacher da wir iterator verwenden
while (iterator.hasNext()) {
    gesamtlänge = gesamtlänge + iterator.next().length();
}

System.out.println("Gesamtlänge: " + gesamtlänge);

// 4. Iterieren Sie mithilfe einer for-each-Loop über myList und geben Sie alle Zeichenketten der Länge 7 aus.
for (String str : myList) {
    if (str.length() == 7) {
        System.out.println(str);
    }
}
}
// -----
}

```

Aufgabe 18a:

```

// Necessary packages/classes:
import java.util.HashMap;
import java.util.Map;
// https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/util/package-summary.html
// https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/util/Map.html
// https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/util/HashMap.html
//

public class Aufgabe_18a {

    // -----

    public static void main(String[] args) {

        // Erstellen Sie ein streng typisiertes HashMap-Objekt für Character-Integer-Paare und eine
        // Variable map, welche dieses Objekt referenziert.
        Map<Character, Integer> map = new HashMap<>();

        // Iterieren Sie über die Kommandozeilenparameter des Programms
        for (String param : args) {
            for (char c : param.toCharArray()) {

                // Berechnen Sie, wie oft jedes enthaltene Zeichen vorkommt
                map.put(c, map.getOrDefault(c, 0) + 1);
            }
        }

        // Geben Sie für alle enthaltenen Zeichen die jeweilige Anzahl der Vorkommen aus
        for (Map.Entry<Character, Integer> entry : map.entrySet()) { alternativ: keySet und map.get(k)
            System.out.println("Zeichen " + entry.getKey() + " kommt " + entry.getValue() + " mal vor.");
        }
    }
}

```

```
// -----
}
```



Aufgabe 18b:

```
// Necessary packages/classes:
import java.util.HashMap;
import java.util.Map;
import java.util.Random;
//

public class Aufgabe_18b {

    // -----

    public static double averageRelativeDeltaFromMean(Map<Integer, Integer> map) {

        // 1. zuerst den Durchschnittswert der Values für alle Keys berechne
        double sum = 0.0;

        for (Integer value : map.values()) {
            sum = sum + value;
        }
        double avg = sum / map.size();

        // 2. für jeden Key die relative absolute Abweichung von diesem Durchschnittswert berechnet
        double sumRelativeDeltas = 0.0;

        for (Integer value : map.values()) {
            sumRelativeDeltas = sumRelativeDeltas + Math.abs(value - avg) / avg;
        }
        // 3. den Durchschnitt dieser relativen Abweichungen für alle Keys berechnet und zurückgibt.
        return sumRelativeDeltas / map.size();
    }

    // -----

    public static void main(String[] args) {

        // 1. erzeugen Sie ein HashMap<Integer, Integer>-Objekt und ...
        Map<Integer, Integer> map = new HashMap<>();

        // ... initialisieren Sie die Werte für die Keys 0 bis 9 mit 0
        for (int i = 0; i < 10; i = i + 1) {
            map.put(i, 0);
        }


        // 2. Berechnen Sie 10 zufällige Zahlen zwischen 0 und 9 und ...
        Random rand = new Random(); // random seed

        for (int i = 0; i < 10; i = i + 1) {
            int num = rand.nextInt(10);

            // ... zählen Sie deren Vorkommen in Ihrer Map
            map.put(num, map.get(num) + 1);
        }

        // 3. Berechnen Sie die durchschnittliche relative Abweichung und geben diese aus
        double avgRelDelta = averageRelativeDeltaFromMean(map);
        System.out.println("Durchschnittliche relative Abweichung: " + avgRelDelta);

        /* 4. Erzeugen Sie dann so lange zufällige Zahlen zwischen 0 und 9, bis die durchschnittliche relative
        Abweichung weniger als 0.01 beträgt */
        while (avgRelDelta >= 0.01) {
```



```

// Generiere Zufallszahl zwischen 0 und 9
int num = rand.nextInt(10);

// Zähle hoch in map
map.put(num, map.get(num) + 1);

// Berechne averageRelativeDelta neu
avgRelDelta = averageRelativeDeltaFromMean(map);
}

// 5. Geben Sie dann die Anzahl der berechneten Zahlen aus
int total = 0;
for (int count : map.values()) {
    total += count;
}
System.out.println("Anzahl der berechneten Zahlen: " + total);
}

// -----
}

```

Aufgabe 19a:

```

// Necessary packages/classes:
import java.util.Arrays;
import java.util.List;
import java.util.function.UnaryOperator;
import java.util.function.Predicate;
//

```

```

public class Aufgabe_19a {

```

```

// -----

```

```

public static void main(String[] args) {

```

```

// 1. eine Liste für String-Objekte erzeugt und diese mit den Kommandozeilenparametern
// befüllt (verwenden Sie dafür eine geeignete Methode der Klasse Arrays),
List<String> list = Arrays.asList(args); // btw: list vs. array: Array is fixed size upon creation

```

```

// 2. ein UnaryOperator-Objekt für Strings erstellt, dass alle Kleinbuchstaben einer übergebenen
// Zeichenkette zu Großbuchstaben konvertiert,
UnaryOperator<String> toUpperCase = String::toUpperCase;

```

/ In Java, UnaryOperator<String> toUpperCase = String::toUpperCase; is a method reference that creates a UnaryOperator<String> which transforms a given string to uppercase.*

UnaryOperator<String> specifies that it's an operation with a single operand of type String, that returns a result of type String.

The :: operator is the method reference operator in Java. It's used to call a method on the input argument. In this case, it calls the toUpperCase method on the instance of String.

So, toUpperCase is a UnaryOperator<String> which, when applied to a string, returns the uppercase version of that string.

**/*

```

// 3. damit alle Zeichenkette in der vorher erstellten Liste transformiert,
list.replaceAll(toUpperCase);

```

```

// 4. mit einem Lambda-Ausdruck für ein Prädikat alle Zeichenketten, die mit "A" beginnen,
// entfernt und
Predicate<String> startsWithA = s -> s.startsWith("A");
list.removeIf(startsWithA);


```

```

// 5. am Ende alle Zeichenketten jeweils in einer eigenen Zeile mithilfe einer Methodenreferenz
// ausgibt.
list.forEach(System.out::println);
}

// -----
}

```



Aufgabe 19b:

```

// Necessary packages/classes:
import java.util.List;
import java.util.ArrayList;
import java.util.function.UnaryOperator;
import java.util.function.Predicate;
import java.util.Random;
//

public class Aufgabe_19b {

    // -----
    public static boolean isPrime(int n) {
        // Smaller equal one = not prime
        if (n <= 1) {
            return false;
        }

        // Loop through all i smaller equal n and test divisibility
        for (int i = 2; i * i <= n; i = i + 1) {

            // Not a prime if modulo is 0
            if (n % i == 0) {
                return false;
            }
        }

        // Return true if above does not apply
        return true;
    }

    // -----

    public static void main(String[] args) {

        // 1. eine Liste für Integer-Objekte erzeugt und diese mit 5000 zufälligen ganzen Zahlen
        // aus dem Intervall [0, 10000[ befüllt,
        Random rand = new Random(); // random seed
        List<Integer> list = new ArrayList<>();
        for (int i = 0; i < 5000; i = i + 1) {
            list.add(rand.nextInt(10000)); // inclusive 0 exclusive 10000
        }

        // 2. direkt mit einem Lambda-Ausdruck alle geraden Zahlen der Liste quadriert,
        UnaryOperator<Integer> squareIfEven = i -> i % 2 == 0 ? i * i : i;
        // if even is true, then square it, if false do nothing
        list.replaceAll(squareIfEven);

        // 3. ein Objekt vom Typ Predicate<Integer> erzeugt, das zurückgibt, ob eine übergebene
        // natürliche Zahl eine Primzahl ist,
        Predicate<Integer> isNotPrime = i -> !isPrime(i);

        // 4. mit diesem Objekt alle Primzahlen aus der Liste entfernt und
        list.removeIf(isNotPrime);
    }
}

```

das sollte ein Predicate<Integer> sein, keine eigene Methode

```
// 5. am Ende alle Zahlen in einer jeweils eigenen Zeile mithilfe einer Methodenreferenz  
//   ausgibt.  
list.forEach(System.out::println);  
}  
  
// -----  
}
```

✓

Σ 4P schön umgesetzt
und kommentiert!
JP