MINT-KOLLEG BADEN-WÜRTTEMBERG

Universität Stuttgart

# Vorkurs Informatik

Dr. Sebastian Müller
Dr. Frank Schweiner

---

## 4. Tutoriumsblatt

Bearbeiten Sie dieses Übungsblatt im Projekt `Day4` und legen Sie für jede Aufgabe ein Paket `task01`, `task02`, ... an.

## Aufgaben Vormittag

### 1. Aufgabe

*Waschbär / Raccoon*

a) Implementieren Sie das in der Vorlesung gezeigte Beispiel mit den Klassen `Test` und `Raccoon`.

b) Schreiben Sie zu jeder Methode der Klasse `Raccoon` einen JavaDoc-Kommentar.

c) Alle Waschbären sind zu Beginn grau. Schreiben Sie eine Methode `setColor(String color)`, mit der die Farbe eines Waschbären nachträglich auf eine andere Farbe `color` geändert werden kann.

d) Ändern Sie die Farbe von Fluffy auf orange.

e) Werden Sie bei der Erweiterung dieses Programmes selbst kreativ.

**Bitte wenden!**

## 2. Aufgabe

*Methoden auf Strings / String Test*

*Hinweis:* Die Klasse String und Ihre Methoden existieren bereits. Sie benötigen hier also nur eine Klasse `Test` mit der `main`-Methode, in der Sie Objekte der Klasse String anlegen.
Verwenden Sie evtl. die Dokumentation der Klasse `String`, um mehr Informationen über die Methoden zu erhalten: https://docs.oracle.com/javase/8/docs/api/java/lang/String.html

a) Erzeugen Sie ein `String`-Objekt mit dem Bezeichner `text` und beliebigem Inhalt.

b) Prüfen Sie mit Hilfe der Methode `boolean isEmpty()` ob `text` leer ist. Falls ja, soll `"Empty text."` ausgegeben werden. Ansonsten soll der Code für den Rest der Aufgabe ausgeführt werden.

c) Ermitteln Sie mit Hilfe von `int length()` die Länge des Textes (Anzahl Zeichen). Geben Sie diese aus.

d) Falls `text` länger als 5 Zeichen ist, geben Sie mit Hilfe von `String substring(int beginIndex, int endIndex)` die ersten fünf Zeichen aus.

e) Wandeln Sie mit der Methode `String toLowerCase()` alle Buchstaben in Kleinbuchstaben um und speichern Sie das Ergebnis in einer Variablen `String lowercaseText`.

f) Verwenden Sie die Methode `boolean startsWith(String prefix)`, um festzustellen ob `lowercaseText` mit der Buchstabenfolge `"das"` anfängt, und geben Sie das Resultat aus.

g) Verwenden Sie die Methode `int indexOf(String str)`, um festzustellen, an welcher Stelle in `lowercaseText` die Buchstabenfolge `"der"` vorkommt, und geben Sie den Index aus. Falls die Folge nicht vorkommt, wird von `indexOf` der Wert -1 zurückgegeben. Geben Sie dann die Meldung `"'der' does not occur in this text."` aus.

h) Prüfen Sie mit Hilfe der Methode `boolean endsWith(String suffix)`, ob `text` mit dem Zeichen `"!"` oder `"."` endet. Wenn das nicht der Fall ist, hängen Sie mit der Methode `String concat(String str)` ein `"."` an `text` an.

Beispielausgabe:

```
Original text: Das IST ein TEXT
Length of this text: 16
First 5 Letters: Das I
Lowercase text: das ist ein text
Starts with 'das': true
'der' does not occur in this text.
```

**Bitte wenden!**

# Aufgaben Nachmittag

## 3. Aufgabe

*Bankkonto / Bank Account*

Laden Sie aus ILIAS die Dateien `Test.java` und `Account.java` herunter. Legen Sie diese in ein Paket `task03`.

*Hinweis:* Achten Sie im Folgenden genau auf die Datentypen und Namen! Sie dürfen die Aufrufe in der Testdatei nicht verändern. Bei nicht zusammenpassenden Aufrufen müssen Sie Ihre Definition der entsprechenden Methode in der Klasse `Person` oder in der Klasse `Konto` verändern.

**Teil 1: Kontoinhaber**

a) Legen Sie eine Klasse `Person` im Paket `task03` an. Jede Person soll einen Namen (`String name`) und ein Alter (`int age`) als Attribute besitzen.

b) Schreiben Sie einen Konstruktor für die Klasse `Person`.

c) Überlegen Sie sich, welche Werte die Attribute nicht annehmen sollten. Stellen Sie im Konstruktor sicher, dass Exceptions geworfen werden, falls versucht wird, ein Objekt mit "illegalen" Werten zu erstellen.

d) Schreiben Sie Getter-Methoden zu beiden Attributen.

e) Schreiben Sie eine Operation `celcebrateBirthday()`. Bei Aufruf soll *`"Let's party!"`* ausgegeben werden und die Person ein Jahr älter werden.

f) Schreiben Sie eine Methode `String toString()`, die *`"(Person: <name>,<age>)"`* als String zurückgibt, wobei die Teile in `<>` durch die Werte der jeweiligen Attribute ersetzt werden sollen. Tipp: `+`.

g) Testen Sie Ihre Klasse, indem Sie die Kommentarzeichen `//` in der Klasse `Test` entfernen.

**Teil 2: Enumeration**

a) Für eine Person soll der Familienstand gespeichert werden. Legen Sie hierfür eine Enumeration `MaritalStatus` an, die folgende Werte enthält: `SINGLE`, `MARRIED`, `DIVORCED`, `WIDOWED`.

b) Ergänzen Sie in der Klasse `Person` ein weiteres Attribut `maritalStatus`, in dem der Familienstand gespeichert wird.

c) Passen Sie den Konstruktor so an, dass jede Person zu Beginn single ist.

d) Ergänzen Sie je eine Getter- und Setter-Methode für das Attribut `maritalStatus`.

**Bitte wenden!**

**Teil 3: Konto - Konstruktoren**

a) Definieren Sie die folgenden Attribute für die Klasse `Account` (alle `private`):

- `String accountNumber` (speichert die Kontonummer)
- `Person owner` (speichert den Kontoinhaber)
- `double balance` (speichert den momentanen Kontostand)

b) Definieren Sie zwei Konstruktoren für die Klasse `Account` (alle `public`):

- Der erste Konstruktor besitzt drei Parameter, sodass alle Attribute initialisiert werden können. Lassen Sie eine Exception werfen, falls versucht wird, ein Konto mit negativem Kontostand zu eröffnen.
- Der zweite Konstruktor besitzt nur zwei Parameter, mit denen die Kontonummer und der Inhaber initialisiert werden können. Der Kontostand wird mit 0.0 initialisiert.

**Teil 4: Konto - Methoden**

a) Definieren Sie die folgenden Methoden der Klasse `Account` (alle `public`):

- `double getBalance()`
  Gibt den momentanen Kontostand zurück.
- `void deposit(double amount)`
  Erhöht den Kontostand um `betrag`, falls `amount` positiv und kleiner als 10000 ist. Ansonsten wird eine Exception geworfen!
- `void withdraw(double amount)`
  Verringert den Kontostand um `amount`, falls `amount` positiv ist und genug Geld auf dem Konto ist. Ansonsten wird eine Exception geworfen!
- `String toString()`
  Gibt `Account No <accountNumber> by <owner>, balance: <balance>`
  als String zurück, wobei die Teile in <> durch die Werte der jeweiligen Attribute ersetzt werden sollen.

**Teil 5: Konto - Überweisungen**

a) Es soll nun die Möglichkeit geschaffen werden, Geld von einem Konto auf ein anderes Konto zu überweisen. Schreiben Sie hierfür eine Methode `transferMoneyTo(double amount, Account account)` inkl. JavaDoc-Kommentar und beachten Sie folgende Punkte:

- Wird auf einem Konto-Objekt - z.B. `paulsAccount` - die Methode aufgerufen, soll sich bei diesem Objekt der Kontostand verringern. Bei dem anderen Konto-Objekt soll sich der Kontostand erhöhen.
- Eine Überweisung soll nur möglich sein, wenn der Betrag positiv ist, wenn das erste Konto nicht überzogen wird und wenn auf das zweite Konto nicht zu viel überwiesen wird (mehr als 10000 Euro).

b) Testen Sie Ihre Methode, indem Sie 10 Euro von Paul an Paula überweisen.

**Bitte wenden!**

# Zusatzaufgaben

## 4. Aufgabe

*Optional*

In der objektorientierten Programmierung steht man öfters vor dem Problem, dass ein Attributwert für ein Objekt fehlt bzw. nicht vorhanden ist. Ein einfaches Beispiel hierfür wäre eine Fax-Adresse: Entwirft man eine Klasse `ContactData` mit Kontaktdaten einer Person, möchte man dafür sorgen, dass prinzipiell eine Fax-Adresse hinterlegt werden kann - aber nicht jede Person besitzt nun einmal eine Fax-Adresse. Abhilfe schafft hier die Klasse `Optional`.

Ein `Optional` kann man sich als Datenbehälter vorstellen kann, der entweder einen Wert enthält oder leer (empty) ist. Folgende vier Methoden werden benötigt:

- `Optional.empty()` legt einen leeren Datenbehälter an
- `Optional.of(..)` legt einen Datenbehälter mit Wert an.
- `boolean isPresent()` liefert zurück, ob der Datenbehälter einen Wert enthält.
- `T get()` liefert den Wert aus dem Datenbehälter vom Typ `T` zurück (sofern vorhanden, andernfalls wirft sie eine `NoSuchElementException`).

Folgendes Beispiel soll die Verwendung der Klasse `Optional` veranschaulichen:

```
public class Test {

   public static void main (String[] args) {

       Adress paulsAddress = new Address("Keplerstrasse 7",70174,"Stuttgart");
       ContactData paulsContactData
           = new ContactData(paulsAddress, "12345", Optional.of("54321"));

       Optional<String> paulsFax = paulsContactData.getFaxNumber();
       if(paulsFax.isPresent()) {
           System.out.println("Paul's fax number is: " + paulsFax.get());
       } else {
           System.out.println("Paul doesnt have a fax number");
       }
   }
}
```

**Bitte wenden!**

```
public class ContactData {

    private Address address;
    private String phoneNumber;
    private Optional<String> faxNumber;

    public ContactData(Address address, String phoneNumber){
        this.address = address;
        this.phoneNumber = phoneNumber;
        this.faxNumber = Optional.empty();
    }

    public ContactData(Address address, String phoneNumber, Optional<String> faxNumber){
        this.address = address;
        this.phoneNumber = phoneNumber;
        this.faxNumber = faxNumber;
    }

    public Address getAddress() {
        return this.address;
    }

    public String getPhoneNumber() {
        return this.phoneNumber;
    }

    public void setFaxNumber(Optional<String> faxNumber) {
        this.faxNumber = faxNumber;
    }

    public Otional<String> getFaxNumber() {
        return this.faxNumber;
    }
}
```

Weitere Informationen finden Sie unter https://docs.oracle.com/javase/8/docs/api/java/util/Optional.html.

**Bitte wenden!**

In dieser Aufgabe möchten wir nun ein Auto mit zwei Plätzen und einem Kofferraum modellieren. Auf jedem der Plätze kann eine Person Platz nehmen. Der Kofferraum kann mit einem Koffer gefüllt werden, sofern dieser nicht zu groß ist.

a) Legen Sie eine Klasse `Suitcase` an, die Koffer mit Höhe `height`, Breite `width` und Länge `length` sowie Füllgewicht `weight` repräsentieren soll. Ein leerer Koffer soll 1kg wiegen. Schreiben Sie den Konstruktor sowie Getter-Methoden und sorgen Sie dafür, dass die Koffermaße nicht negativ sein dürfen.
Schreiben Sie zudem eine Methode `fill(int weight)` zum Füllen des Koffers. Der Koffer wird dadurch natürlich schwerer. Außerdem soll es eine Methode `empty()` zum Leeren des Koffers geben.

b) Kopieren Sie die Klasse `Person` aus der Aufgabe mit dem Bankkonto. Ergänzen Sie die Klasse um ein Attribut `weight`. Passen Sie den Konstruktor entsprechend an und ergänzen Sie eine Getter-Methode.

c) Nun erstellen wir die Klasse `SmartCar`. Ein Auto besitze ein bestimmtes Leergewicht `emptyWeight`. Weiterhin können sich optional bis zu zwei Personen (`driver` und `coDriver`) und optional ein Koffer `suitcase` im Auto befinden. Schreiben Sie auch hier einen Konstruktor (der ein leeres Auto mit einem Leergewicht von mind. 800kg erzeugt) sowie Getter-Methoden und Setter-Methoden (aber keine Setter-Methode für das Leergewicht). Ein Koffer kann nur verladen werden, wenn seine Maße $80 \times 90 \times 45$ nicht überschreiten.

d) Ergänzen Sie in der Klasse `SmartCar` eine Operation, die das Gesamtgewicht des Autos berechnet und den Wert anschließend zurückgibt.

e) Legen Sie eine Klasse `Test` mit der `main`-Operation an. In dieser instanziieren Sie zwei Personen-Objekte (Paul und Paula, jeweils mit 80kg) sowie einen Koffer der Größe $20 \times 80 \times 40$, der anschließend mit 20kg Gepäck gefüllt wird. Instanziieren Sie nun ein `SmartCar` mit Leergewicht von einer Tonne. Laden Sie die Personen sowie den Koffer ins Auto ein und lassen Sie sich anschließend das Gesamtgewicht ausgeben. Anschließend soll der Beifahrersitz wieder geleert werden.

f) Wenn alles geklappt hat, testen Sie zum Schluss einige Fehlerfälle, d.h. überprüfen Sie, ob bei folgenden Szenarien Exceptions geworfen werden:

- Erstellen einer Person mit $-4$kg.
- Erstellen eines Koffers mit den Maßen $0 \times 20 \times 40$.
- Füllen des Koffers mit $-4$kg.
- Verladen eines Koffers mit den Maßen $90 \times 90 \times 90$ ins Auto.