Programmierung 1

im Wintersemester 2025/26

Praktikumsaufgabe 2

Klassen und Objekte (Kap. 2)

Review: 11. November 2025 (DFIW) und 12. November 2025 (PI)

Mit diesen Aufgaben tauchen Sie in die objektorientierte Programmierung ein. In Aufgabe 1 setzen Sie Schleifen und Bedingungen ein, um die Kontrolle über den Programmfluss zu erlangen. In Aufgabe 2 entwerfen Sie eine Klasse, einer der entscheidenden Schritte in der objektorientierten Programmierung. In Aufgabe 3 erstellen Sie Objekte einer vorgegebenen Klasse, während Sie in Aufgabe 4 Ihre eigene Klasse entwerfen, implementieren und testen.

Heads up!

Für diese Praktikumsaufgabe – *wie auch für die folgenden* – gilt:

OneCompiler: Verwenden Sie den OneCompiler für die Lösung und speichern Sie Ihren Code dort. Auf diese Weise können Sie den Code während des Reviews einfach im Browser präsentieren.

Links: Die Aufgaben enthalten an einigen Stellen Links zu weiterführenden Informationen. Seien Sie neugierig und folgen Sie diesen Links, um mehr über die Themen zu erfahren.

Wie bei der ersten Praktikumsaufgabe gilt weiterhin: Lösen Sie die Aufgaben nicht alleine, sondern mindestens zu zweit! Zum Beispiel mit Ihrer Coding-Partner:in. Denn in der Gruppe lernt es sich besser.

Wir als Ihre Lernbegleiter:innen unterstützen Sie gerne bei der Lösung der Aufgaben.

1 Schleifen und Bedingungen

Lernziele

- ☐ Sie fragen Eingaben von einer Benutzer:in ab und geben Ausgaben an diese aus.
- ☐ Sie verwenden **`for`**- und **`while`**-Schleifen, um eine Aktion mehrmals auszuführen.
- ☐ Sie verwenden **`if`**-/**`else`**-Tests, um verschiedene Anweisungen in Abhängigkeit von einer Bedingung auszuführen.
- ☐ Sie verwenden verschachtelte Kontrollstrukturen, um kompliziertere Abläufe zu steuern.

Mit dieser Aufgabe üben Sie die Kontrolle über den Programmfluss (engl. control flow) durch *Schleifen* und *Bedingungen* aus. Um das Programm interessanter zu gestalten, verwenden Sie *Benutzereingaben und -ausgaben*: Stellen Sie sich vor, Sie erstellen eine ausgeklügelte Begrüßungsfunktion für eine clevere App.

Sie können das folgende Programm als Ausgangspunkt für Ihre Lösung verwenden:

```java
// Mithilfe des Import-Statements können wir auf vordefinierte Klassen
// aus anderen Paketen zugreifen. Hier importieren wir das Paket
// `java.util`, das nützliche Klassen wie `Scanner` für die Eingabe
// über die Konsole enthält. Mehr unter:
// https://www.baeldung.com/java-packages
import java.util.*;

public class ControlFlowTestDrive {

  public static void main(String[] args) {
    Scanner inputScanner = new Scanner(System.in);

    System.out.println("Geben Sie den Namen des Ortes ein, in dem Sie
     ↪  wohnen: ");
    String residence = inputScanner.nextLine();
    System.out.println("In " + residence + " ist es bestimmt sehr
     ↪  schön.");

    inputScanner.close();
  }
}
```

In den folgenden Aufgaben können Sie den Code entsprechend anpassen.

Heads up!

Bei kleinen Programmen wie diesem ist es gute Praxis, Ressourcen wie den `Scanner` am Ende mit `close()` wieder freizugeben.

Profi-Tipp: In großen Anwendungen ist Vorsicht geboten! Ein `Scanner`, der `System.in` liest, schließt mit `close()` auch den dahinterliegenden Eingabestrom des gesamten Programms – und zwar dauerhaft. Das kann später zu unerwarteten Fehlern führen, wenn ein anderer Teil des Programms ebenfalls auf eine Eingabe wartet. Für den Moment ist unser Vorgehen aber genau richtig!

1.1 Eingaben und Ausgaben

Fragen Sie die Benutzer:in nach ihrem Namen und begrüßen Sie sie mit diesem Namen!

- Der Text der Aufforderung macht deutlich, dass der eigene Name eingegeben werden soll.
- Der Name wird mit einer freundlichen Begrüßung ausgegeben.

Heads up!

Aussagekräftige Benennung: Die saubere ("cleane") Benennung von Variablen (Klassen, Methoden, ...) ist entscheidend für die Lesbarkeit und Wartbarkeit des Codes und für die Minimierung von Fehlern im Code, da sie Zweck und Inhalt klar kommuniziert. Gut benannte Variablen sind beschreibend, konsistent und vermeiden Mehrdeutigkeiten, was die Zusammenarbeit im Entwicklerteam erleichtert.

Englische Namen: Englische Namen für Variablen (Klassen, Methoden, ...) sind gängige Praxis, da sie die internationale Verständlichkeit fördern, technische Komplikationen vermeiden und die Konsistenz des Codes gewährleisten. Dies erleichtert die Wartung und die Zusammenarbeit in multikulturellen Entwicklerteams.

Wählen Sie also *immer* einen Namen für Ihre Variablen, der *diesen Konventionen entspricht* und damit zeigt, dass Sie auf dem Weg zu einer professionellen Programmierer:in sind.

1.2 Schleifen

Begrüßen Sie die Benutzer:in mehrmals!

- Verwenden Sie eine **`for`**- oder **`while`**-Schleife, um die Begrüßung mehrmals auszugeben. So können Sie beispielsweise die Benutzer:in gleich dreimal begrüßen.
- Passen Sie dann Ihren Code so an, dass aus der **`for`**-Schleife eine **`while`**-Schleife wird oder umgekehrt. Die Ausgabe soll sich nicht ändern.

1.3 Bedingungen

Überprüfen Sie, ob die Benutzer:in einen Namen eingegeben hat!

- Erweitern Sie Aufgabe 1.2 so, dass die Begrüßung nach Eingabe eines Namens weiterhin mehrfach ausgegeben wird.
- Hat die Benutzer:in nichts eingegeben, so soll eine freundliche Fehlermeldung ausgegeben werden.
- Ansonsten soll die wiederholte Begrüßung mit dem eingegebenen Namen ausgegeben werden.

Hinweis: Verwenden Sie die Methode `isEmpty()` der Klasse `String` in Ihrer Testbedingung.[1] Sie können beispielsweise wie folgt prüfen, ob ein Wohnort eingegeben wurde:

```
if (residence.isEmpty()) { ... }
```

Wenn Sie einen Schritt weiter gehen möchten, können Sie auch prüfen, ob eine Eingabe nur aus Leerzeichen besteht. Verwenden Sie dazu die Methode `trim()`, wie hier gezeigt:[2]

```
if (residence.trim().isEmpty()) { ... }
```

[1] Mit dieser Methode können Sie prüfen, ob ein String leer ist.
[2] Diese Methode entfernt führende und nachfolgende Leerzeichen aus einem String.

2 Entwurf von Klassen

Lernziele

- ☐ Sie entwerfen eine Klasse in Pseudo-UML, indem Sie deren Zustand (Attribute) und Verhalten (Methoden) identifizieren.
- ☐ Sie erläutern den Sinn und Zweck von Klassen als Baupläne für Objekte.

Strecklernziele

- ☐ Sie modellieren eine Klassenhierarchie, indem Sie Unterklassen entwerfen, die von einer Superklasse erben.
- ☐ Sie begründen die Vorteile von Vererbung zur Wiederverwendung und Spezialisierung von Code.

In dieser Aufgabe entwerfen Sie *Klassen*, die einen Bauplan für Objekte mit ihren *Instanzvariablen* und *Methoden* darstellen. Eine Klasse beschreibt, was ein Objekt in Form von Instanzvariablen "weiß" (auch *Attribute* genannt) und was ein Objekt in Form von Methoden "tun" kann. Der Entwurf einer Klasse ist einer der ersten Schritte auf dem Weg zu einem professionellen Java-Programm und vermittelt ein grundlegendes Verständnis der objektorientierten Programmierung.

Achtung! Verwenden Sie für Ihre Entwürfe *Pseudo-UML*, wie es im Buch "Java von Kopf bis Fuß" praktiziert wird. Darin finden Sie auch Beispiele für Pseudo-UML-Diagramme.

2.1 Klasse

Entwerfen Sie eine Klasse `Person`, die grundlegende Informationen über eine Person speichert und eine einfache Begrüßung ausgeben kann!

2.1.1 Entwurf: `Person`

Beschreiben Sie die Attribute und Methoden Ihrer Klasse in *Pseudo-UML*. Das heißt, Sie erstellen eine Visualisierung der Klasse `Person`, die deren Attribute und Methoden darstellt,

bevor Sie die Klasse in Java implementieren. An dieser Stelle wird daher kein Java-Code, sondern ein UML-Diagramm erwartet.

Überlegen Sie, was eine Person weiß und was eine Person tun kann. Zum Beispiel könnte eine Person folgendes wissen und tun:

- Eine Person kennt ihren Namen.
- Eine Person weiß, wie alt sie ist.
- Eine Person kann grüßen und sich dabei vorstellen.[3]

In diesem Beispiel repräsentieren der Name und das Alter das Wissen über eine Person, das Grüßen repräsentiert ihr Tun. Wählen Sie Attribute und Methoden, die Ihrer Meinung nach eine Person beschreiben, und es ihr erlauben, sich vorzustellen.

Heads up!

Wie Sie bereits wissen, erleichtert eine Benennung nach den Konventionen der professionellen Programmierung die Lesbarkeit und Wartbarkeit erheblich. Dies gilt auch für Attribute und Methoden im Klassendesign. *Achten* Sie also *schon beim Entwurf* darauf.

2.1.2 Reflexion: Sinn und Zweck von Klassen

Reflektieren Sie abschließend, warum es sinnvoll ist, Attribute und Methoden in einer Klasse zusammenzufassen. Überlegen Sie dabei, wie eine Klasse `Person` als Bauplan für beliebig viele Personen (Objekte) dienen kann.

2.2 Vererbung: Superklasse und Unterklasse

Erweitern Sie den Entwurf der Klasse `Person` um die beiden Unterklassen `Student` und `Professor`, die jeweils spezielles Wissen und Tun repräsentieren!

[3] `"Hallo, ich heiße Jessica!"`, `"Hallo, ich heiße Joe!"`, oder ...

Die neuen Klassen verdeutlichen das Konzept der Vererbung in der objektorientierten Programmierung.

2.2.1 Entwurf: `Student`

Zunächst zur neuen Klasse `Student`. Überlegen Sie, welches zusätzliche oder spezialisierte Wissen und Tun für eine Student:in sinnvoll ist.

Dies könnte beispielsweise sein:

- Eine Student:in weiß, in welchem Studiengang sie eingeschrieben ist.
- Eine Student:in kennt ihre Matrikelnummer.
- Eine Student:in kann grüßen und bei dieser Gelegenheit gleich den Studiengang nennen.[4]
- Eine Student:in kann ihre Matrikelnummer nennen.

Der Studiengang und die Matrikelnummer repräsentieren hier das zusätzliche Wissen über eine Student:in. Das Grüßen als Student:in mit gleichzeitiger Nennung des Studiengangs repräsentiert das *spezialisierte* Tun, die Nennung der Matrikelnummer das *zusätzliche* Tun. Sie können Attribute und Methoden auswählen, die Ihrer Meinung nach eine Student:in beschreiben und ihr erlauben, sich als Student:in vorzustellen.

2.2.2 Entwurf: `Professor`

Schließlich zur neuen Klasse `Professor`. Überlegen Sie, welches zusätzliche oder spezialisierte Wissen und Tun für eine Professor:in sinnvoll ist.

[4] `"Hallo, ich heiße Jessica und studiere Praktische Informatik!"`, oder ...

2.2.3 Reflexion: Sinn und Zweck von Vererbung

Reflektieren Sie abschließend die Vorteile der Vererbung gegenüber der wiederholten Implementierung von Attributen und Methoden. Überlegen Sie dabei, warum es sinnvoll sein kann, eigene Klassen wie `Student` und `Professor` zu erstellen, anstatt alle Attribute und Methoden in `Person` zu integrieren. Beachten Sie, dass das Entwerfen von Unterklassen es ermöglicht, gemeinsame Merkmale zusammenzufassen und Spezialisierungen effizient abzubilden.

3 Erstellen und Nutzen von Objekten

Lernziele

- ☐ Sie erzeugen mit dem **`new`**-Operator Objekte (Instanzen) einer gegebenen Klasse.
- ☐ Sie greifen mit dem Punktoperator auf Attribute und Methoden eines Objekts zu, um dessen Zustand zu manipulieren und dessen Verhalten aufzurufen.
- ☐ Sie erkennen, dass Objekte derselben Klasse eigenständige Instanzen mit einem unabhängigen Zustand sind.
- ☐ Sie schreiben eine Testklasse, um die Funktionalität einer anderen Klasse zu demonstrieren.

In dieser Aufgabe erstellen Sie Objekte einer Klasse. Sie greifen auf deren Attribute zu und rufen deren Methoden auf.

3.1 Objekte der Klasse `Person`

Erstellen Sie dazu eine eigene, separate Testklasse `PersonTestDrive`, die dies an der Klasse `Person` durchspielt!

Verwenden Sie die folgende beispielhafte Implementierung der Klasse Person als Ausgangspunkt für Ihre Lösung:

```
1 class Person {
2   int age;
3   String name;
4
5   void greet() {
6     System.out.println("Hallo, ich heiße " + name + "!");
7   }
8 }
```

Im Folgenden können Sie diesen Code direkt verwenden. Erstellen Sie dazu im OneCompiler über das Pluszeichen eine neue Java-Datei (*New Java file*) und geben Sie den Code dort

ein. Vergessen Sie nicht, die Datei in `Person.java` umzubenennen.[5] Fahren Sie dazu mit dem Mauszeiger über den Namen der Klasse (`NewClass1.java`), dann erscheint das Bleistiftsymbol, über das Sie einen anderen Namen vergeben können.

Dies gibt Ihnen die Möglichkeit, die Klasse `Person` in der Praxis zu verwenden und zu testen, wie Objekte erstellt und ihre Attribute und Methoden verwendet werden.

3.1.1 Testklasse `PersonTestDrive`

Implementieren Sie eine Testklasse `PersonTestDrive`, die in ihrer `main`-Methode ein Objekt der Klasse `Person` erzeugt, seine Attribute initialisiert und seine Methode aufruft. Führen Sie die folgenden Schritte aus:

1. Erstellen Sie mit **`new`** ein neues Objekt der Klasse `Person`.

2. Verwenden Sie den Punktoperator (`.`), um den Namen und das Alter der Person festzulegen, ...

3. ... und um ihre Methode `greet()` aufzurufen.

Hinweis: Wenn Sie bereits mit OO vertraut sind, werden Sie feststellen, dass wir hier nicht kapseln, wie es eigentlich der professionellen Praxis entspricht. Wir vereinfachen hier bewusst und werden in den nächsten Wochen darauf zurückkommen.

3.1.2 Mehrere Objekte erstellen und nutzen

Erstellen Sie nun mehrere Objekte der Klasse `Person` und setzen Sie deren Attribute auf unterschiedliche Werte. Rufen Sie dann für jedes Objekt die Methode `greet()` auf und beobachten Sie, wie sich die Ausgabe entsprechend ändert.

Hinweis: Der Einfachheit halber können Sie die einzelnen Personen in etwas generisch klingenden Variablen wie `person1`, `person2`, ... speichern:

[5] Vereinfacht gesagt muss eine Klasse in einer Datei gespeichert werden, die genau den gleichen Namen wie die Klasse hat, einschließlich Groß- und Kleinschreibung. Die Datei muss die Endung `.java` haben.

```java
Person person1 = new Person();
// ...
Person person2 = new Person();
// ...
```

3.2 Bonusaufgaben

Hinweis: Diese Bonusaufgaben können Sie beispielsweise lösen, wenn Sie bereits programmieren können und deshalb die anderen Aufgaben schnell gelöst haben. Oder wenn Sie einfach Spaß am Programmieren gefunden haben und sich weiter ausprobieren möchten.

3.2.1 Benutzerdefinierte Namen

Nutzen Sie Ihr Können aus Aufgabe 1.1, um die Benutzer:in nach ihrem Namen zu fragen und diesen Namen für Objekte der Klasse `Person` zu verwenden. Auf diese Weise können Sie die Methode `greet()` personalisieren und Ihre Testklasse `PersonTestDrive` interaktiver gestalten.

3.2.2 Benutzerdefiniertes Alter

Gehen Sie einen Schritt weiter und fragen Sie die Benutzer:in nach Namen und Alter. Verwenden Sie diese Eingaben, um ein Objekt der Klasse `Person` zu initialisieren. Sie können dann zum Beispiel die Begrüßung so oft ausgeben, wie es dem in dem Objekt gespeicherten Alter entspricht.

Tipp: Mit der Methode `inputScanner.nextInt()` kann eine Ganzzahl, z. B. das Alter, eingelesen werden. Der Scanner liest die Eingabe aus dem *STDIN*-Feld des OneCompilers, das zeilenweise ausgelesen werden kann. Wenn Sie `nextLine()` gefolgt von `nextInt()` aufrufen, können Sie so zuerst den Namen (als `String`) und dann das Alter (als `int`) einlesen.

4 Erstellen und Nutzen von Klassen

Lernziele

- ☐ Sie wenden die erlernten Konzepte durch Transfer auf eine neue Problemstellung an.
- ☐ Sie überführen einen Klassenentwurf in Form von Pseudo-UML in eine lauffähige Java-Implementierung.
- ☐ Sie implementieren eine Testklasse, um die korrekte Funktionalität Ihrer selbst entwickelten Klasse zu überprüfen und zu demonstrieren.

In dieser Aufgabe entwickeln Sie eigenständig eine Klasse und eine dazugehörige Testklasse. Diese Aufgabe fordert Sie heraus, die Prinzipien der Klassenentwicklung selbst anzuwenden. Dabei wenden Sie Ihre bisher erworbenen Kenntnisse über Klassen, Attribute und Methoden in der Programmierpraxis an.

Die Aufgabe orientiert sich an den Klassen `Dog` und `DogTestDrive` aus dem Buch. Hier entwickeln Sie jedoch eine Klasse `Cat` mit einer Testklasse `CatTestDrive`. Ziel ist es, durch Transfer selbstständig ein vollständiges Objektmodell zu entwerfen und zu testen. Dabei können Sie sich am konkreten Beispiel orientieren.

Heads up!

Halten Sie sich als angehende professionelle Programmierer:in *an die Ihnen nun bekannten Konventionen!* Dazu gehören insbesondere die aussagekräftige Benennung von Klassen, Attributen, Methoden und Variablen sowie die Trennung zwischen der “echten” Klasse, deren Objekte Sie tatsächlich verwenden wollen, und der Testklasse, deren Zweck allein darin besteht, die Funktionalität der Klasse zu überprüfen bzw. zu demonstrieren.

4.1 Entwurf: `Cat`

Entwerfen Sie Ihre eigene Klasse `Cat`, die die Eigenschaften und das Verhalten einer typischen Katze durch geeignete Attribute und Methoden beschreibt!

Die Klasse soll mindestens zwei Attribute und eine Methode enthalten. Die Methode soll ein typisches Katzengeräusch erzeugen (ausgeben). Verwenden Sie Pseudo-UML für Ihren Entwurf.

Tipp: Während ein Hund bellt (engl. bark), miaut eine Katze (engl. meow).[6] Da Katzen eitler sind als Hunde, haben sie kein Alter, sondern eine Fellfarbe (engl. coat color). Und Katzen interessieren sich nicht für ihre Rasse (engl. breed), sondern für ihre verbleibenden Leben (engl. remaining lives).

4.2 Klasse: `Cat`

Implementieren Sie die Klasse `Cat`!

Ihre Implementierung soll die Attribute und Methoden enthalten, die Sie zuvor entworfen haben. Die Methode soll das für Ihre Katze typische Geräusch ausgeben.

4.3 Klasse: `CatTestDrive`

Schreiben Sie eine Testklasse `CatTestDrive` für Ihre Klasse `Cat`!

Erstellen Sie in der `main`-Methode ein Objekt der Klasse `Cat`, legen Sie die Attribute fest und rufen Sie die Methode auf, um das für Ihre Katze typische Geräusch auszugeben.

Nachdem Sie sich auf diese Weise vergewissert haben, dass Sie Ihre Klasse korrekt implementiert haben, erzeugen Sie weitere Objekte der Klasse `Cat`. Geben Sie für jede Katze ihre Eigenschaften (Attribute) aus und lassen Sie sie das Katzengeräusch ausgeben. Zum Beispiel:

```
1 Cat cat1 = new Cat();
2 cat1.name = "Carlson";
3 cat1.coatColor = "black";
4 cat1.remainingLives = 7;
5 System.out.print(cat1.name + ", the cat");
6 System.out.print(" with " + cat1.coatColor + " fur");
7 System.out.println(" and " + cat1.remainingLives + " lives
  ↪  left, says:");
```

[6] Eine englischsprachige Katze würde `"Meow!"` machen, eine deutschsprachige `"Miau!"`.

```
8  cat1.meow();
9  Cat cat2 = new Cat();
10 // ...
```