

# Programmieren - Übungen

Matthias Berg-Neels

*[Download Übungen](#)*

# Kapitelübersicht - Programmieren 1

1. Einführung
2. Grundlagen von Java
3. Datentypen
4. Ausdrücke und Anweisungen
5. Objektorientierung
6. Vererbung
7. Interfaces

# Kapitelübersicht - Programmieren 2

- 8. Exception Handling
- 9. Collection Framework
- 10. Swing
- 11. Optional: Input- & Output-Stream
- 12. Datenstrukturen
- 13. Algorithmen

# Kapitel 1

## Einführung

Kontrollfragen & Übungen

## 1.1 Kontrollfragen

1. Nennen Sie die Definition für den Begriff des Algorithmus im Kontext der Informatik.
2. Beschreiben Sie die Bestandteile sowie die Eigenschaften von Algorithmen.
3. Nennen Sie fünf Grundelemente der Programmierung.
4. Welche Möglichkeiten stehen Ihnen zur Verfügung, um Algorithmen grafisch darzustellen?
5. Was ist ein wesentlicher Nachteil der Programmablaufpläne?
6. Beschreiben Sie die Darstellungsform „Pseudocode“!

## 1.2 Übungen

1. Beschreiben Sie den euklidischen Algorithmus zur Berechnung des größten gemeinsamen Teilers als Struktogramm und Programmablaufplan!
2. Beschreiben Sie einen Algorithmus zur Berechnung der Fakultät! Nutzen Sie dazu den Pseudocode.
3. Beschreiben Sie im Pseudocode einen Algorithmus zur Bestimmung der Fibonacci Folge 1 – 1 – 2 – 3 – 5 – 8 – 13 – 21!
4. Beschreiben Sie in einem Struktogramm das Sieb des Eratosthenes zur Bestimmung von Primzahlen.

# Kapitel 2

# Grundlagen von Java

## Kontrollfragen

## 2.1 Kontrollfragen

1. Beschreiben Sie die wesentlichen Eigenschaften von Java!
2. Beschreiben Sie die Funktionsweise eines Compilers!
3. Welche Aufgaben übernimmt der Linker?
4. Nennen Sie unterschiedliche Interpreter-Arten!
5. Welche Rolle spielen Compiler und Interpreter im Umfeld der Programmiersprache Java?
6. Beschreiben Sie den Prozess von der Erstellung des Quellcodes bis zur Ausführung des Programms in der Programmiersprache Java!
7. Nennen Sie drei wesentliche Java-Tools und beschreiben Sie kurz deren Aufgaben!
8. Beschreiben Sie das Paketkonzept der Programmiersprache Java!
9. Welche wesentlichen Systemvariablen kennen Sie im Java-Umfeld?
10. Beschreiben Sie die wesentlichen Unterschiede der verschiedenen Editionen im Rahmen der Java 2™ Platform!
11. Wozu werden die einzelnen Systemvariablen benötigt?



# Kapitel 3

# Datentypen

## Kontrollfragen

## 3.1 Kontrollfragen

1. Welche unterschiedlichen Datentypen kennen Sie in Java?
2. Wie lassen sich diese Datentypen klassifizieren?
3. Nennen Sie die numerischen Datentypen der Programmiersprache Java! Worin liegt der wesentliche Unterschied im Wertebereich?
4. Wie können Variablen und Konstanten in Java deklariert initialisiert werden?
5. Was ist der Unterschied zwischen einer Konstanten und einer Variablen?
6. Erläutern Sie den Begriff „Literal“ am Beispiel der numerischen Literale!
7. Was sind Escape-Sequenzen?
8. Beschreiben Sie die Konvertierungsregeln in Java bzgl. Erweiternder und einschränkender Konvertierungen für einfache Datentypen!
9. Was ist ein Array und wie ist dieses aufgebaut?
10. Welche Aufgabe erfüllt das Attribut length bei einem Array?
11. Worin unterscheidet sich die Deklaration eines Arrays von der Deklaration einer einfachen Variablen?
12. Was sind Referenzdatentypen?

# Kapitel 4

## Ausdrücke und Anweisungen

### Kontrollfragen & Übungen

## 4.1 Kontrollfragen

1. Was verstehen Sie im Zusammenhang mit Programmierung unter dem Begriff „Ausdruck“?
2. Woraus setzen sich Ausdrücke zusammen?
3. Zählen Sie unterschiedliche Operatoren auf! Unterscheiden Sie dabei die Operatoren nach dem Typ der Operanden!
4. Erläutern Sie den Fragezeichenoperator anhand des folgenden Codings!

```
String X = (a == b) ? "Ja" : "Nein";
```

5. Definieren Sie den Begriff „Anweisung“ im Sinne der Programmierung!
6. Welche Möglichkeiten bietet Ihnen die Programmiersprache Java, um Verzweigungen zu realisieren?
7. Erläutern Sie den Begriff „dangling else“!
8. Welche Bedeutung messen Sie der Break-Anweisung im Zusammenhang mit der Switch-Anweisung bei?
9. Nennen Sie die unterschiedlichen Schleifenarten in Java und beschreiben Sie deren Verhalten!
10. Worin besteht der wesentliche Unterschied zwischen einer kopf- und einer fußgesteuerten Schleife?
11. Beschreiben Sie den Schleifenkopf einer For-Schleife!
12. Was bewirken die Break- und die Continue-Anweisung innerhalb einer Schleife?

## 4.2 Übungen

1. Implementieren Sie Ihr Struktogramm aus Übung 1 im Kapitel 1. zur Berechnung des größten gemeinsamen Teilers nach Euklid!
2. Implementieren Sie Ihre Pseudocode-Lösung aus Übung 2 im Kapitel 1. zur Berechnung der Fakultät einer Zahl!
3. Implementieren Sie Ihr Struktogramm aus Übung 4 im Kapitel 1. zur Bestimmung der Primzahlen nach dem Sieb des Eratosthenes, wobei der Anwender eine beliebige Zahl eingeben soll. Verwenden Sie dazu folgende Eingabemöglichkeit:

```
import javax.swing.JOptionPane;  
//...  
String s = JOptionPane.showInputDialog("Geben Sie eine Zahl ein:");  
int zahl = Integer.parseInt(s);
```

4. Formulieren Sie eine Anweisung, die zur folgenden semantisch äquivalent ist zu folgender Anweisung ohne Verwendung des Wortes "if" und unter Verwendung logischer Operatoren.

```
return a==b ? false : true;
```

## 4.2 Übungen

5. Gegeben sind folgende Variablen mit ihren Werten:

```
int a = 10;  
int b = 5;  
boolean z = false;
```

Füllen Sie folgende Wahrheitstabelle aus:

Ausdruck	Wahrheitswert
!z	
a < 20	
a == 2 * b	
a % b != 0	
(a > b) && z	
(a > b)    z	
!(a < b) ^ !z	
(a < b)    ((a % 3 < b) && !z)	

## 4.2 Übungen

6. Folgende Ausgabe soll erzeugt werden:

```
1 3 5 7 9
```

Lösen Sie diese Aufgabe mithilfe einer

- While-Schleife
- Do-while-Schleife
- For-Schleife und des Modulo-Operators
- For-Schleife ohne den Modulo-Operator

Lösen Sie die Teilaufgaben zunächst mit einem Struktogramm, bevor Sie mit der Implementierung beginnen.

7. Folgende Ausgabe soll erzeugt werden:

```
1 2 4 7 11 16 22 29 37
```

Lösen Sie diese Aufgabe mithilfe einer

- Do-while-Schleife
- For-Schleife

## 4.2 Übungen

8. Entwerfen Sie eine For-Schleife, die die Buchstaben A bis Z auf dem Bildschirm ausgibt.
9. Welche mathematischen Operationen werden durch die Methoden `meth1()` und `meth2()` beschrieben und welche Werte haben `zahl1` und `zahl2` am Ende der Berechnung?

```
public class TestAlgorithmen {
    public static void main(String[] args) {
        int zahl1 = meth1(-5);
        int zahl2 = meth2(2, 6);
        System.out.println("Zahl 1:\t" + zahl1 + "\nZahl 2:\t" + zahl2);
    }
    static int meth1(int a) {
        if(a < 0)
            a=-a;
        int rc=0;
        for( int i=0; i<a; i++ )
            rc += a;
        return rc;
    }
    static int meth2(int a, int b) {
        // Wenn b<0 wird eine Fehlermeldung erzeugt -> 2. Semester
        // Die Fehlermeldung führt zum Abbruch der Methode
        if(b < 0)
            throw new RuntimeException( "unerlaubter Wert");
        if(b == 0)
            return 1;
        int rc = a;
        for(int i = 1; i < b; i++)
            rc *= a;
        return rc;
    }
}
```



## 4.2 Übungen

10. Setzen Sie Ihre Lösung zur Bestimmung der Fibonacci-Folge aus Übung 3 im Kapitel 1.2 in ein Programm um, wobei die Zahlenfolge dauerhaft in einem Array gespeichert werden soll!
11. Beschreiben Sie einen iterativen Algorithmus zur Berechnung der Wurzel einer Zahl nach dem Heron-Verfahren (babylonisches Wurzelziehen) als Struktogramm! Dabei gilt folgende Berechnungsvorschrift:  $a$  = Zahl, deren Quadratwurzel bestimmt werden soll  $X(n)$  = Ergebnis des letzten Iterationsschrittes, wobei  $X(0)$  dem Wert  $a$  entspricht

$$X(n + 1) = \frac{X(n) + \frac{a}{X(n)}}{2}$$

Die Berechnung soll abbrechen, sobald die Differenz zwischen  $X(n+1)$  und  $X(n)$  weniger als  $\pm 0,000001$  beträgt.

## 4.2 Übungen

12. Beschreiben Sie in einem Struktogramm einen Algorithmus zur Berechnung des Pascal'schen Dreiecks und implementieren Sie diesen.

Beispiel:

```
1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
1 5 10 10 5 1
1 6 15 20 15 6 1
1 7 21 35 35 21 7 1
```

# Kapitel 5

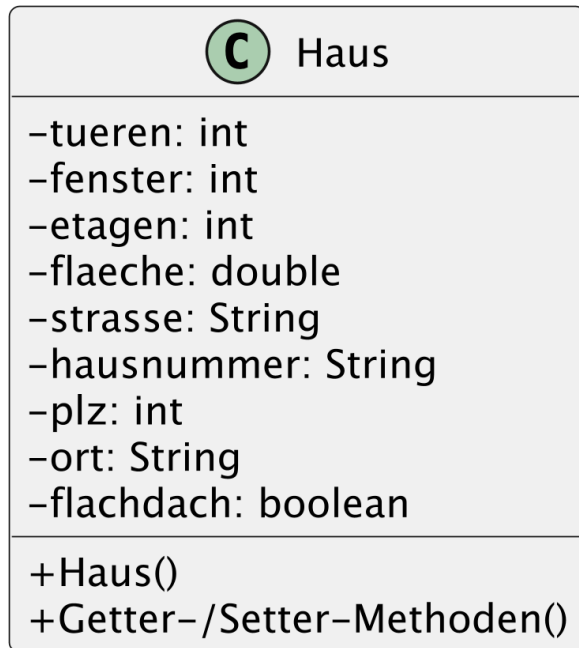
# Objektorientierung

## 5.1 Kontrollfragen

1. Grenzen Sie die Begriffe Klasse und Objekt voneinander ab!
2. Was beschreiben Attribute und Methoden?
3. Wie werden Klassen in der UML dargestellt?
4. Wie werden in Java Objekte erzeugt? Welche Rolle spielen dabei die Konstruktoren einer Klasse?
5. Erläutern Sie den Begriff des Konstruktors!
6. Beschreiben Sie das Konzept der Kapselung in der objektorientierten Programmierung!
7. Welche unterschiedlichen Sichtbarkeitsmodifizier sind Ihnen bekannt? Beschreiben Sie die Sichtbarkeit jedes Modifiers!
8. Beschreiben Sie den Zugriff auf Instanzattribute und -methoden! Welche Rolle spielen dabei die Sichtbarkeitsmodifizier?
9. Unterscheiden Sie die Begriffe „call by value“ und „call by reference“!
10. Was versteht man unter dem Begriff „überladen von Methoden“?
11. Worin unterscheiden sich Klassenattribute und -methoden von Instanzattributen und -methoden?
12. Die Klasse `java.lang.Math` stellt eine Sammlung von mathematischen Standardfunktionen dar, die allesamt als `static` definiert sind. Weil die Klasse über keine Instanzmethoden oder -variablen verfügt, wäre eine Erzeugung von Objekten dieser Klasse recht unsinnig. Um dies zu verhindern, haben ihre Programmierer einen Trick angewendet. Wie konnten sie eine Instanziierung verhindern, ohne die Klasse abstrakt zu definieren?
13. Wie können Klassenmethoden auf den Instanzenkontext zugreifen?
14. Erläutern Sie das Prinzip der Garbage Collection in Java!
15. Was sind Destruktoren und wie werden Sie aufgerufen?
16. Erläutern Sie die Begriffe Assoziation, Aggregation und Komposition! Wie werden die unterschiedlichen Beziehungen in der UML dargestellt?

## 5.2 Übungen - Haus (1/2)

- Gegeben ist folgendes UML-Diagramm einer Klasse Haus:

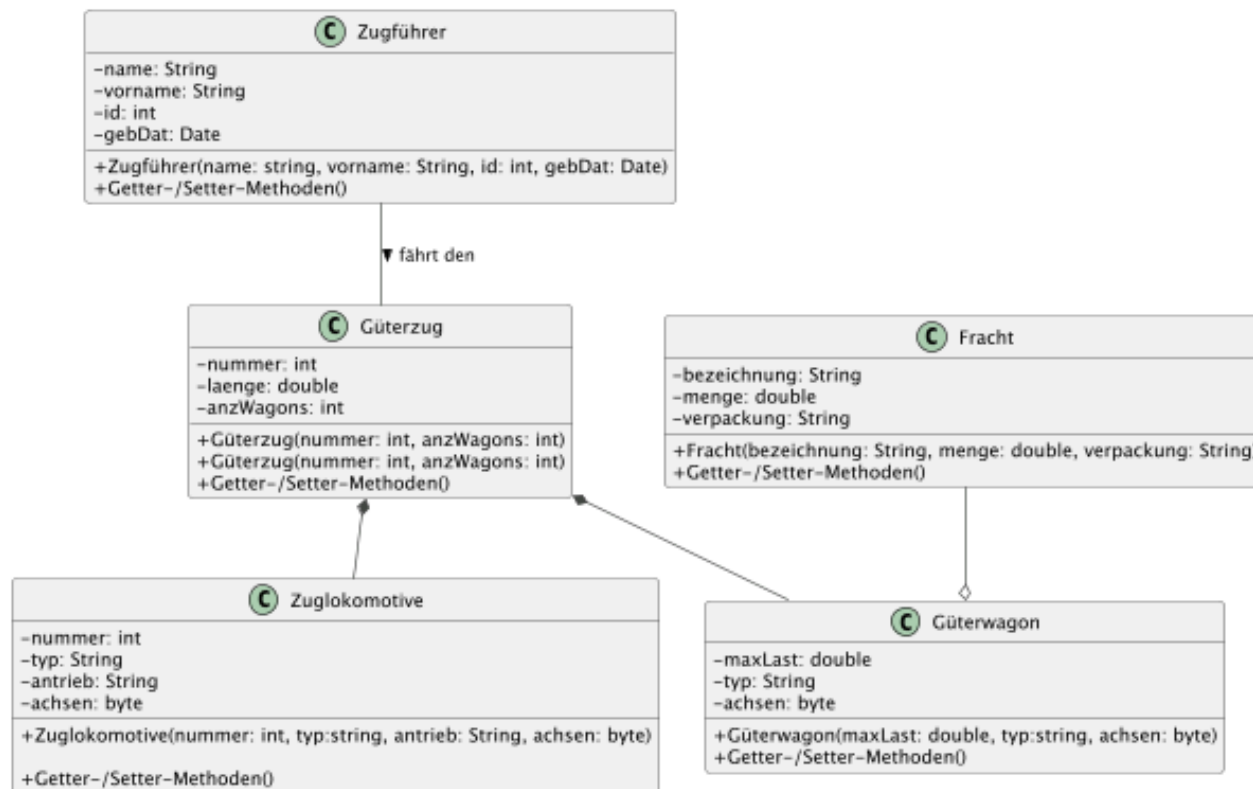


## 5.2 Übungen - Haus (2/2)

- Übung 1
  - Implementieren Sie die Klasse Haus unter folgenden Voraussetzungen:
    - Bei den Setter-Methoden soll die Plausibilität der übergebenen Werte überprüft werden (z.B. keine leeren Strings, keine negativen Postleitzahlen, keine negative Anzahl an Türen, Fenstern oder Etagen, eine minimale Wohnfläche größer 100 und kleiner 500, usw.). --> Werden keine gültigen Werte übergeben, sollen selbstgewählte Default-Werte genutzt werden.
    - Schreiben Sie zusätzlich eine Testklasse HausTest, in der Sie 3 Hausobjekte erzeugen und in einem Array speichern. Geben Sie alle Hauseigenschaften der im Array gespeicherten Hausobjekte innerhalb einer For-Schleife aus.
- Übung 2
  - Erweitern Sie die Klasse Haus um weitere Konstruktoren. Es soll zu jedem Attribut ein eigener Konstruktor zur Verfügung gestellt werden. Außerdem soll ein Konstruktor zur Verfügung stehen, der alle Attribute entgegen nimmt.
- Übung 3
  - Erweitern Sie die Klasse Haus aus Übung 1 um ein statisches Attribut objCnt vom Typ Integer mit dem Initialwert 0 und die statische Methode getObjCnt() vom Typ Integer. Ändern Sie Ihren Konstruktor so ab, dass objCnt um 1 erhöht wird, sobald ein neues Objekt erzeugt wird. Geben Sie in der Klasse HausTest nach der Ausgabe der Eigenschaften der Hausobjekte die Anzahl der erzeugten Objekte aus.
- Übung 4
  - Erweitern Sie Ihre Klasse Haus um die finalize()-Methode. Dekrementieren Sie objCnt um 1, wenn ein Objekt gelöscht wird. Löschen Sie zum Test am Ende Ihrer Klasse HausTest ein.

## 5.2 Übungen - Bahnhof (1/2)

- Gegeben ist das folgende Diagramm, das die Assoziationen, Aggregationen und Kompositionen zwischen den Klassen abbildet.



## 5.2 Übungen - Bahnhof (2/2)

- Übung 1
  - Nennen Sie die Beziehungsarten, die zwischen den Klassen vorherrschen.
- Übung 2
  - Wie können die Beziehungen sinnvoll realisiert werden? Erweitern Sie die nachstehenden Klassendiagramme um fehlenden Attribute oder Methoden!
- Übung 3
  - Implementieren Sie die einzelnen Klassen und berücksichtigen Sie dabei die Beziehungsarten.
- Übung 4
  - Implementieren Sie eine Testklasse `Bahnhof`, um einen Güterzug zu erstellen, ihm einen Fahrer zuzuordnen und die Güterwagons mit Fracht zu beladen. Geben Sie anschließend über die jeweiligen Getter-Methoden die Informationen zum Fahrer, dem Güterzug und der geladenen Fracht aus.



## 5.2 Übungen - Smartphone

### 1. Übung

- Erstellen Sie ein Klassendiagramm für die Klasse Smartphone (und möglicher weiterer notwendigen Klassen) nach folgenden Anforderungen
  - Attribute:
    - brand (Marke) – String.
    - model (Modell) – String.
    - price (Preis) – Double.
    - type (Typ) – SmartphoneType (Werte: ANDROID, IOS)
  - Methoden:
    - Konstruktor mit allen Attributen
    - Getter und Setter für alle Attribute

### 2. Übung

- Implementieren Sie die Klassen Smartphone
  - Implementieren Sie die Setter-Methoden mit Plausibilitätsprüfungen:
    - brand und model dürfen keine leeren Strings sein. Falls ungültige Werte übergeben werden, setzen Sie "Unknown"
    - type darf nicht null sein und soll auf einen passenden Standardwert gesetzt werden
    - price darf nicht negativ sein. Bei ungültigen Werten wird der Standardwert 0.0 gesetzt

### 3. Übung

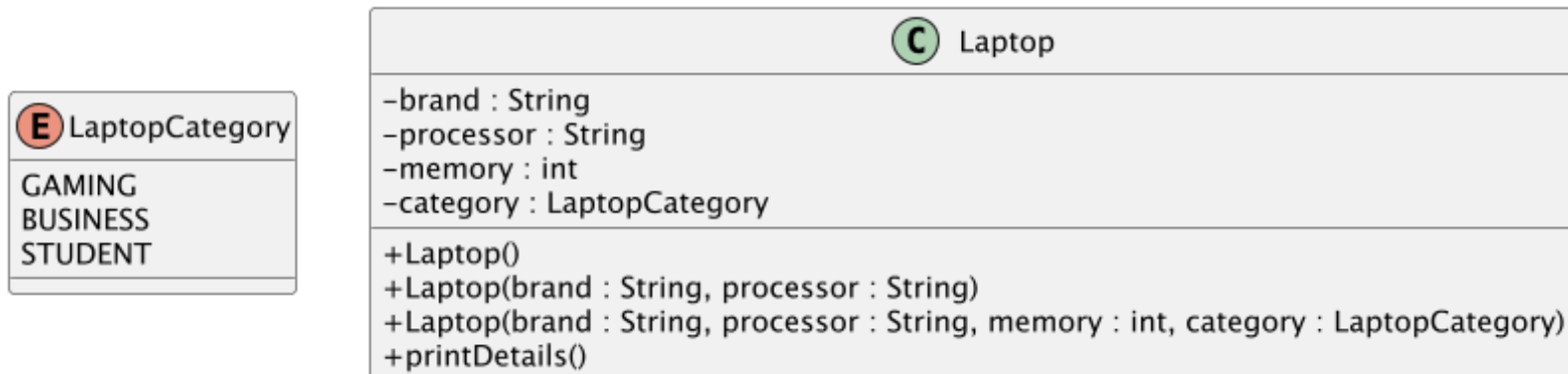
- Fügen Sie die Methode printDetails() hinzu, die die Eigenschaften des Smartphones ausgibt
  - Überladen Sie die Methode, sodass optional eine zusätzliche Nachricht (z. B. "Special offer!") ausgegeben werden kann
  - Erweitern Sie hierzu auch das Klassendiagramm

### 4. Übung

- Schreiben Sie eine Testklasse SmartphoneTest
  - Erzeugen Sie drei Smartphone-Objekte mit unterschiedlichen Werten
  - Rufen Sie die Methode printDetails() für jedes Smartphone auf – einmal ohne Nachricht und einmal mit einer Nachricht

## 5.2 Übungen - Laptop

- Gegeben ist folgendes UML-Diagramm einer Klasse Laptop:



### 1. Übung

- Implementieren Sie die Klasse `Laptop` und weitere notwendige Klassen
  - Implementieren Sie Konstruktoren:
    - Einen Standard-Konstruktor, der `brand` und `processor` auf "Unknown" setzt und `memory` auf 0
    - Einen Konstruktor, der `brand` und `processor` entgegennimmt
    - Einen Konstruktor, der alle Attribute initialisiert
  - Implementieren Sie die Methode `printDetails()` die alle Attribute ausgibt

### 2. Übung

- Schreiben Sie eine Testklasse `LaptopTest`:
  - Erzeugen Sie mindestens drei `Laptop`-Objekte mit verschiedenen Konstruktoren
  - Geben Sie die Details aller Laptops aus

## 5.2 Übungen - Book

### 1. Übung

- Erstellen Sie ein Klassendiagramm mit der Klasse Book für folgende Anforderungen:
  - title (Titel) – String.
  - author (Autor) – String.
  - pages (Seitenanzahl) – Integer.
  - price (Preis) – Double.
- Fügen Sie ein statisches Attribut bookCount und eine entsprechende Methode zum Auslesen dieses Attributs hinzu
  - Dieses Attribut soll die Anzahl der erstellten Bücher mitzählen

### 2. Übung

- Implementieren Sie die Klasse Book

### 3. Übung

- Erweitern Sie das Klassendiagramm um eine Methode printDetails(), die die Attribute eines Buchs ausgibt, und implementieren Sie diese

### 4. Übung

- Schreiben Sie eine Testklasse BookTest:
  - Erzeugen Sie mehrere Buch-Objekte
  - Geben Sie die Details aller Bücher aus
  - Geben Sie die Gesamtanzahl der erstellten Bücher mithilfe von bookCount aus

# Kapitel 6

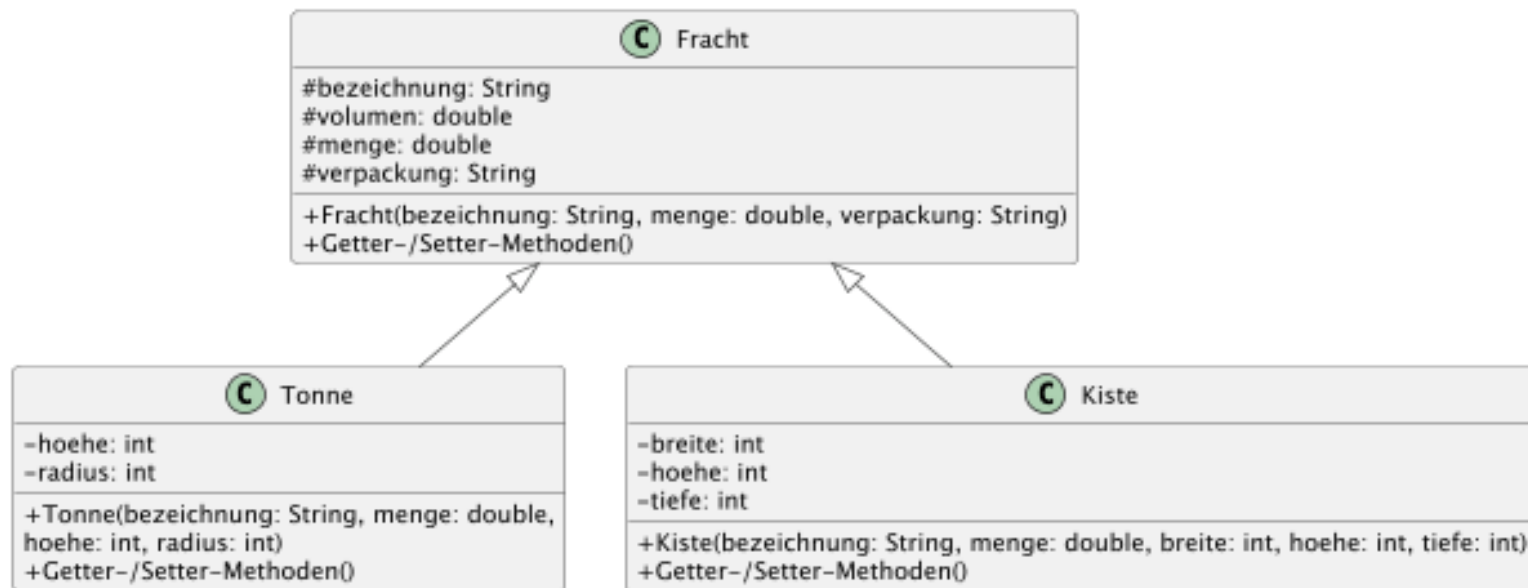
# Vererbung

# 6.1 Kontrollfragen

1. Beschreiben Sie das objektorientierte Konzept der Vererbung.
2. Erläutern Sie die Begriffe Superklasse, Subklasse und Vererbungshierarchie.
3. Beschreiben Sie die Darstellung der Vererbung in der UML mithilfe eines kleinen Beispiels.
4. Was ist die Besonderheit in der Vererbung bei Java und wie ist die Vererbung syntaktisch in Java realisiert?
5. Beschreiben Sie die wesentlichen Eigenschaften der Klasse Object und ihre besondere Rolle in der Vererbungshierarchie in Java?
6. Beschreiben Sie das Konzept des Überschreibens von Methoden!
7. Was bewirken die Modifier `abstract` und `final` in Bezug auf Klassen?
8. Welche Auswirkungen haben die Modifier `abstract` und `final` bei Methoden?
9. Wozu wird der Ausdruck `super` in der Vererbung benötigt?
10. Was stellt der Ausdruck `this` dar?
11. Beschreiben Sie das Konzept des `narrowing` und `widening Cast`!
12. Beschreiben Sie das Konzept des Polymorphismus in der objektorientierten Programmierung anhand eines einfachen Beispiels!
13. Welche Möglichkeiten haben Sie, um in Java Objekte zu kopieren? Worin besteht der Unterschied in den beiden Verfahren?
14. Innerhalb eines Pakets `mypackage` werden zwei Klassen `Vater` und `Sohn` definiert. `Sohn` ist eine Subklasse der Klasse `Vater`. Beide verfügen über eine Methode `familienbande()`, die nur für Kindklassen innerhalb des Pakets zugänglich sein soll.
  - Welcher Modifikator ist also für die Methode zu nehmen: `public`, `protected`, `private` oder der Standardmodifikator?

## 6.2 Übungen - Bahnhof (1/2)

- Das nachstehende Vererbungsdiagramm erweitert das Modell aus der Übung 5 aus dem Übungsteil 5.2 Bahnhof.



## 6.2 Übungen - Bahnhof (2/2)

- Übung 1
  - Erweitern Sie die Klasse Fracht um das Attribut Volumen und die entsprechende Getter-Methode für dieses Attribut. Implementieren Sie anschließend die beiden Subklassen Kiste und Tonne.
- Übung 2
  - Beim Aufruf der Konstruktoren der Sub-Klassen soll zunächst der Konstruktor der Super-Klasse gerufen werden, um die allgemeinen Attribute zu setzen. Die Bezeichnung der Verpackung soll dabei als konstanter Wert „Kiste“ oder „Tonne“ gesetzt werden. Berechnen Sie innerhalb des Konstruktors das Volumen der Fracht.
- Übung 3
  - In den Subklassen sollen lediglich die Getter- und Setter-Methoden für die subklassenspezifischen Attribute zur Verfügung gestellt werden.
- Übung 3
  - Erweitern Sie Ihre Testklasse Bahnhof aus der Übung 5. aus dem Übungsteil 5.2., indem Sie unterschiedliche Objekte der Klassen Kiste und Tonne erzeugen und als Fracht in die Güterwagons laden. Nutzen Sie dazu das Konzept der Polymorphie. Geben Sie die Informationen zu Ihren Frachtobjekten über eine polymorphe Implementierung aus.

# Kapitel 7

# Interfaces

Kontrollfragen & Übungen



## 7.1 Kontrollfragen

1. Was ist ein Interface in Java, und warum wird es verwendet?
2. Wie unterscheiden sich Interfaces von abstrakten Klassen in Java?
3. Was passiert, wenn eine Klasse ein Interface implementiert, aber nicht alle Methoden definiert?
4. Kann ein Interface Konstanten enthalten? Wenn ja, wie werden sie definiert?
5. Wie kann eine Klasse mehrere Interfaces implementieren, und was passiert bei Namenskonflikten?
6. Können Interfaces andere Interfaces erweitern? Wenn ja, wie funktioniert das?
7. Was passiert, wenn eine Klasse ein Interface implementiert, das Methoden mit der gleichen Signatur wie eine geerbte Klasse definiert?
8. Kann eine abstrakte Klasse ein Interface implementieren, ohne dessen Methoden zu definieren?
9. Kann eine Klasse ein Interface und eine Klasse gleichzeitig erben?
10. Können Interfaces Konstruktoren enthalten?

## 7.2 Übungen - Smarthome (1/2)

Entwickeln Sie ein Programm zur einfachen und zentralen Steuerung eines Smarthomes. Ziel ist es, eine zentrale Verwaltung für alle steuerbaren Geräte zu schaffen. Dazu sollen folgende **Anforderungen** umgesetzt werden:

### 1. Das Interface `HomeControllable`:

- Das Interface `HomeControllable` soll die grundlegende Steuerung der Geräte ermöglichen:
- `turnOn()`: Schaltet das Gerät ein.
- `turnOff()`: Schaltet das Gerät aus.

### 2. Die Haushaltsgeräte:

- Es werden für den Anfang vier Haushaltsgerätetypen angeboten:
- `CeilingLamp` (Deckenlampe):
  - Diese Klasse soll ein Attribut `room` besitzen, das den Raum angibt, in dem sich die Lampe befindet (z. B. "Wohnzimmer", "Schlafzimmer"). Dieses Attribut wird über den Konstruktor befüllt.
  - Die Methoden `turnOn()` und `turnOff()` sollen Ausgaben, entsprechend der jeweiligen Methode, wie "Die Lampe im Wohnzimmer wurde angeschaltet" enthalten, wobei der Raumname dynamisch ist.
- `VacuumCleaner` (Staubsauger):
  - Gibt beim Einschalten "Der Staubsauger wurde angeschaltet." und beim Ausschalten "Der Staubsauger wurde ausgeschaltet" aus.
- `CoffeeMachine` (Kaffeemaschine):
  - Gibt beim Einschalten "Die Kaffeemaschine wurde angeschaltet." und beim Ausschalten "Die Kaffeemaschine wurde ausgeschaltet." aus.
- `Television` (Fernseher):
  - Gibt beim Einschalten "Der Fernseher wurde angeschaltet." und beim Ausschalten "Der Fernseher wurde ausgeschaltet." aus.

### 3. Zentrale Steuerung: `HomeManager`:

- Die Klasse `HomeManager` ist für die zentrale Verwaltung der Geräte verantwortlich
- Die Methode `shutdown()` soll alle Geräte im Haushalt ausschalten

## 7.2 Übungen - Smarthome (2/2)

Setzen Sie die folgenden **Aufgaben** um:

### 1. Übung

- Erstellen Sie ein Klassendiagramm passend zu den Anforderungen.

### 2. Übung

- Implementieren Sie das Interface, die Haushaltsgeräte und die Klasse HomeManager

### 3. Übung

- Interaktive Simulation:
  - Erstellen Sie eine Main-Methode in der Klasse HomeManager in der mehrere Haushaltsgeräte erzeugt werden
  - Demonstrieren Sie das Einschalten aller Geräte, gefolgt von der zentralen Steuerung durch den HomeManager, der alle Geräte ausschaltet.

### 4. Übung

- **Zusatzaufgabe:** Ändern Sie die Klasse CeilingLamp so ab, dass nur vorbestimmte Werte im Attribut room angegeben werden können. Die Vorgegebene Werte sollen Englischbenannt sein, aber in der Ausgabe sollen die entsprechenden Übersetzten Begriffe auf Deutsch in den Ausgaben stehen. Bei Bedarf erweitern Sie hierzu auch das Klassendiagramm. (Wiederholung zu Kapitel 5 & 6)

# Kapitel 8

# Exception Handling

## Kontrollfragen & Übungen

## 8.1 Kontrollfragen

1. Welche Unterschiedlichen Fehler kennen Sie im Java-Umfeld? Welche Fehler sollten nicht, können oder müssen behandelt werden?
2. Welche Arten von Ausnahmen sind Ihnen im Java-Umfeld bekannt?
3. Erläutern Sie das Grundprinzip der Ausnahmebehandlung!
4. Was verstehen Sie unter „eine Ausnahme“ werfen, fangen und weitergeben?
5. Welches sind die wesentlichen Methoden der Klasse Throwable?
6. Worin unterscheiden sich checked und unchecked Exceptions?
7. Wozu dient der Finally()-Block bei einer Try-Catch-Anweisung?
8. Welche Möglichkeiten sind Ihnen zum Erzeugen eigener Ausnahmeklassen bekannt?
9. Was würde passieren, wenn eine Ausnahme nicht abgefangen, sondern immer weitergegeben wird?

## 8.2 Übungen

1. Das folgende Programm soll die Zahlen von 1 bis 100 in eine Textdatei mit dem Namen `ausgabe.txt` schreiben. Leider kann das Programm in dieser Form nicht ausgeführt werden. Was ist der Grund dafür und wie können Sie es korrigieren? (Anmerkung: Auch wenn der Input- & Output-Stream noch nicht behandelt wurde, können Sie die Frage schon jetzt beantworten.)

```
import java.io.FileWriter;

public class Uebung1 {
    public static void main(String[] args) {
        FileWriter datei;
        String text;

        datei = new FileWriter("ausgabe.txt");
        text = "1\n";

        for(int i = 2; i <=100; i++){
            text += i;
            text += "\n";
        }


        datei.write(text, 0, text.length());
        datei.flush(); }
}
```

## 8.2 Übungen

2. In Ihrem Unternehmen beziehen Sie Autositze. Für das Beziehen stehen Ihnen die Materialien Leder und Stoff zur Verfügung. Den Stoff können Sie in jeder beliebigen Farbe liefern. Leder jedoch ist nur in den Farben Schwarz und Weiß lieferbar.

Implementieren Sie eine Klasse `AutoSitze` gemäß der Vorgaben des UML- Diagramms. Der Konstruktor soll eine Fehlermeldung der von Ihnen zu implementierenden Fehlerklasse `FalscheParameter` erzeugen, sobald er ungültige Parameterkombinationen erhält. Der Meldungstext der Fehlerklasse soll die fehlerhafte Parameterkombination ausgeben.

Testen Sie Ihre Klasse `AutoSitze` mit einem kleinen Testprogramm `TestAutoSitzeException`. In Ihrem Testprogramm soll nach erfolgreichem Durchlauf des Konstruktors eine Meldung ausgegeben werden, in welcher Farbe und in welchem Material der Sitz bezogen wurde. Sollte während dem Durchlauf des Konstruktors eine Ausnahme ausgelöst werden, geben Sie bitte in Ihrem Testprogramm den Meldungstext der Ausnahme sowie einen kleinen Hinweis, dass das Beziehen fehlgeschlagen ist, aus.

 <code>AutoSitze</code>
<code>-ledersitz: boolean</code> <code>-farbe: String</code>
<code>+AutoSitze (bezug: boolean, farbe: String)</code> <code>+getFarbe(): String</code> <code>+istLedersitz(): boolean</code>

## 8.2 Übungen

3. Im folgenden Quellcode befindet sich ein logischer Fehler. Worin besteht er und wie kann er behoben werden?

```
public class TankLeerDemo {  
    public static void main(String[] args) {  
        Auto bmw = new Auto(0, 35487); bmw.tanken();  
  
        bmw.tanken();  
        try {  
            bmw.fahren();  
        } catch (TankLeer e) {  
            System.out.println(e.getMessage());  
        }  
  
        bmw.tanken();  
  
        try {  
            bmw.fahren();  
        } catch (Exception e) {  
            e.printStackTrace();  
        } catch (TankLeer e) {  
            System.out.println(e.getMessage());  
            System.out.println(e.toString());  
            e.printStackTrace();  
        } finally {  
            System.out.println("Der neue Kilometerstand: " +  
                               bmw.getKmCount());  
            bmw.getKmCount();  
        }  
    }  
}
```



## 8.3 Übungen

Nur eines der folgenden Quellcodesnippets (Snippet 1 - 7) funktioniert korrekt?

Snippet 1:

```
import java. io.*;
public class Snippet1 {
    public static void main (String[] args) {
        FileReader f = new FileReader ("Snippet1.java");
        while (true) {
            int c=f. read ();
            if (c<0)
                return;
            System.out.print ((char)c);
        }
    }
}
```

Snippet 2:

```
import java.io.*;
public class Snippet2 {
    public static void main(String[] args) {
        FileReader f = new FileReader("Snippet2.java");
        try {
            while (true) {
                int c=f.read();
                if (c<0)
                    return;
                System.out.print((char)c);
            }
        } catch(IOException e,FileNotFoundException f) {}
    }
}
```

## 8.3 Übungen

### Snippet 3:

```
import java.io.*;

public class Snippet3 {

    public static void main(String[] args) {

        FileReader f = new FileReader("Snippet3.java");
        try {
            while (true) {
                int c=f.read();
                if (c<0)
                    return;
                System.out.print((char)c);
            }
        } catch(FileNotFoundException e) {}
        catch(IOException e) {}
    }
}
```

### Snippet 4:

```
import java.io.*;

public class Snippet4 {

    public static void main(String[] args) throws FileNotFoundException,IOException {

        FileReader f = new FileReader("Snippet4.java");
        try {
            while (true) {
                int c=f.read();
                if (c<0)
                    return;
                System.out.print((char)c);
            }
        }
    }
}
```

## 8.3 Übungen

### Snippet 5:

```
import java.io.*;

public class Snippet5 {

    public static void main(String[] args) throws FileNotFoundException {

        try {
            FileReader f = new FileReader("Snippet5.java");
            while (true) {
                int c=f.read();
                if (c<0)
                    return;
                System.out.print((char)c);
            }
        } catch(IOException e) {}
        } catch(FileNotFoundException e) {}
    }
}
```

### Snippet 6:

```
import java.io.*;

public class Snippet6 {

    public static void main(String[] args) throws FileNotFoundException {

        FileReader f = new FileReader("Snippet6.java");
        try {
            while (true) {
                int c=f.read();
                if (c<0)
                    return;
                System.out.print((char)c);
            }
        } catch(IOException e) {}
    }
}
```

## 8.3 Übungen

### Snippet 7:

```
import java.io.*;

public class Snippet7 {
    public static void main(String[] args) {
        {
            FileReader f=new FileReader("Snippet7.java");
            while (true) {
                int c=f.read();
                if (c<0)
                    return;
                System.out.print((char)c);
            }
        } catch(IOException e) {}
    }
}
```

# Kapitel 9

# Collection Framework

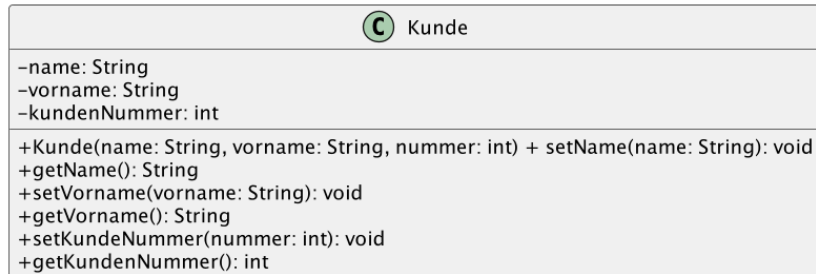
## Kontrollfragen & Übungen

## 9.1 Kontrollfragen

1. Was können Datencontainer des Collection Framework enthalten?
2. Welche drei Arten von Containern kennen Sie?
3. Worin unterscheiden sich die drei Containerarten?
4. Was ist der Unterschied zwischen Arrays und Containern des Collection Framework?
5. Sie möchten ein Schachfeld oder auch den Spielplan von „Schiffe versenken“ in Java implementieren. Dazu müssen Sie einen Repräsentanten des Spielfeldes im Speicher erzeugen und verwalten. Welche Datenstrukturen würden Sie dabei verwenden? Wie begründen Sie Ihre Entscheidung?
6. Wozu werden Iteratoren benötigt?
7. Was müssen Sie bei Containern beachten, in denen die Objekte sortiert abgelegt werden?
8. Wozu brauchen Sie die Interfaces Comparable und Comparator?
9. Können Sie Listen-Container sortieren? Wenn ja, wie gehen Sie bei der Realisierung vor?
10. Welche Möglichkeiten haben Sie, um Objekte miteinander zu vergleichen? Worin liegt der Unterschied der Vergleichsmöglichkeiten?
11. Was müssen Sie beim Überschreiben der equals()-Methode für direkte und indirekte Sub-Klassen der Klasse Object beachten?
12. Worin besteht der Zusammenhang zwischen der hashCode()- und der equals()- Methode?
13. Beschreiben Sie zwei einfache Möglichkeiten, um eine hashCode()-Methode zu überschreiben!
14. Was sind Wrapper-Klassen und wozu werden sie benötigt?
15. Sie legen Objekte der Wrapper-Klasse Double in einem Datencontainer der Klasse TreeSet ab. Warum ist dies problemlos möglich?
16. Worin liegen die wesentlichen Unterschiede zwischen Maps und Lists?

## 9.2 Übungen

1. Implementieren Sie zunächst eine Klasse Kunde gemäß der Vorgaben des UML- Diagramms.



Schreiben Sie sich ein Testprogramm `TestKunde`. In diesem Testprogramm erzeugen Sie sich zunächst fünf Objekte der Klasse `Kunde`. Namen, Vornamen und Kundennummern finden Sie in dieser Tabelle:

Name	Vorname	Kundennummer
Mustermann	Klaus	4711
Beispiel	Hans	5180
Mustermann	Hilde	4712
Vorbild	Theodor	8278
Dummy	Jimmy	1111

Nachdem Sie die Objekte erzeugt haben, möchten Sie die Objekte in einem Datencontainer der Klasse `TreeSet` ablegen. Erweitern Sie entsprechend Ihre Klasse `Kunde`, so dass die Objekte der Klasse `Kunde` aufsteigend nach der Kundennummer sortiert werden.

Geben Sie abschließend die Elemente aus dem `TreeSet` über einen Iterator auf der Konsole aus, um den Erfolg der Sortierung zu überprüfen.

## 9.2 Übungen

2. Erweitern Sie die Klassen `Kunde` und `TestKunde` aus der vorangegangenen Übung so, dass Sie die Objekte der Klasse `Kunde` in einem Datencontainer der Klasse `Vector` nach dem Namen und bei Namensgleichheit nach dem Vornamen aufsteigend sortiert ablegen können.

Geben Sie abschließend auch die Elemente aus dem `Vector` über einen `Iterator` ebenfalls auf der Konsole aus, um auch den Erfolg der zweiten Sortierung zu überprüfen.

3. Ermöglichen Sie es, dass Objekte der Klasse `Kunde` aus den vorangegangenen Übungen miteinander verglichen werden können. Implementieren Sie dazu alle erforderlichen Methoden in der Klasse `Kunde`.

Testen Sie die Vergleichsmethode(n), indem Sie in einem Testprogramm `TestVergleichKunde` zwei gleiche Objekte der Klasse `Kunde` erzeugen und diese miteinander vergleichen. Geben Sie das Ergebnis des Vergleichs auf der Konsole aus.



# Kapitel 10

## Swing

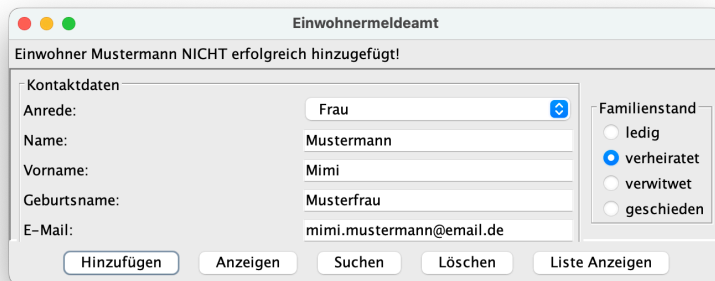
Kontrollfragen & Übungen

## 10.1 Kontrollfragen

1. Erläutern Sie die fünf wesentlichen Unterschiede zwischen AWT- und Swing- Komponenten!
2. Beschreiben Sie die wesentlichen Komponenten eines JFrame-Fensters!
3. Welche Bedeutung kommt den Layout-Managern bei der Gestaltung von Benutzeroberflächen zu?
4. Erläutern Sie den Zusammenhang zwischen Containern der Klasse JPanel und den Layout-Managern!
5. Welche unterschiedlichen Rahmenarten sind Ihnen bekannt?
6. Nennen Sie fünf Swing-Komponenten und deren Funktion zur Gestaltung von Benutzerdialogen!
7. Welche unterschiedlichen Arten von Textfeldern sind Ihnen bekannt?
8. Beschreiben Sie die Vorgehensweise bei der Erstellung von validierenden Textfeldern!
9. Beschreiben Sie die Aufgaben und Anwendungsgebiete des ItemListener!
10. Wozu wird der ActionListener benötigt? Wie funktioniert er?
11. Beschreiben Sie die wesentlichen Unterschiede zwischen den Objekten der Klassen JCheckBox und JRadioButton!
12. Welche unterschiedlichen Klassen und Interfaces benötigen Sie zum Erzeugen von Benutzermenüs? Beschreiben Sie jeweils mit einem Satz die entsprechenden Klassen!
13. Was sind Tooltips und wie werden diese in Java realisiert?

## 10.2 Übungen


1. Entwickeln Sie einen Benutzerdialog zum Erfassen von Einwohnern in der Klasse `EinwohnerMeldeamtUI` analog des Screenshots.



Stellen Sie dabei zunächst nur die Eingabefelder für die Kontaktdaten, die Radiobuttons für den Familienstand und die Buttons zur Verfügung. Dabei ist zu beachten, dass die Auswahlliste für die Anrede die Werte Herr, Frau und Divers zur Verfügung stellen soll. Die Buttons sollen noch keine Funktionalität beinhalten, sondern nur angezeigt werden. Der Nachrichtentext (im Screenshot „Einwohner hinzugefügt“) wird erst zu einem späteren Zeitpunkt implementiert. Beim Schließen des Fensters soll die Anwendung beendet werden.

## 10.2 Übungen

2. Entwickeln Sie die Klasse `Einwohner`. Die Klasse `Einwohner` soll das Interface `Comparable` implementieren. Stellen Sie einen Konstruktor und die entsprechenden Getter-Methoden für die privaten Attribute zur Verfügung. Die `compareTo`-Methode soll Objekte nach dem Namen und bei gleichen Namen nach dem Vornamen aufsteigend sortieren.

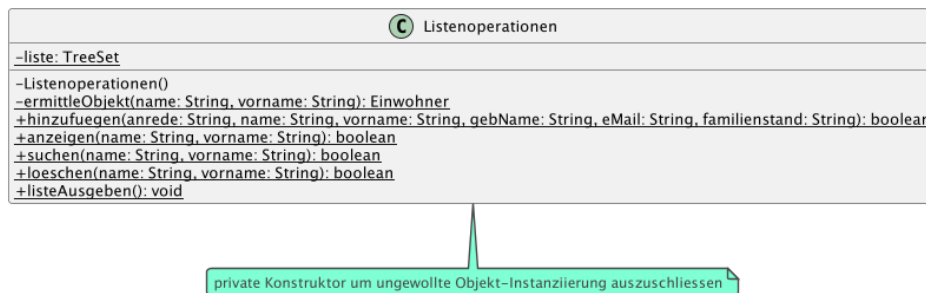
 Einwohner
<ul style="list-style-type: none"><li>-name: String</li><li>-vorname: String</li><li>-gebName: String</li><li>-eMail: String</li><li>-anrede: String</li><li>-familienstand: String</li></ul>
<ul style="list-style-type: none"><li>+Einwohner(name: String, vorname: String, gebName: String, eMail: String, anrede: String, familienstand: String)</li><li>+compareTo(einwohner: Einwohner): int</li><li>+Getter/Setter für privat Attribute()</li></ul>

Über die Klasse `Einwohner` sollen später Einwohner-Objekte auf Basis der im Benutzerdialog gemachten Eingaben erzeugt und in einem `TreeSet` gespeichert werden.

## 10.2 Übungen

3. Stellen Sie für die Radiobuttons und die Buttons je einen eigenen ActionListener zur Verfügung.

- Der ActionListener für die Radiobuttons soll folgende Funktionalität erfüllen. Wählt der Benutzer den Familienstand „ledig“ aus, so wird der Beschreibungstext für den Geburtsnamen ausgeblendet und das zugehörige Eingabefeld initialisiert und ausgeblendet. Wird ein anderer Familienstand angeklickt, so sollen die beiden Felder wieder eingeblendet werden.
- Die Methoden für die Umsetzung der Funktionalität sollen als Klassenmethoden in einer eigenen Klasse `Listenoperationen` zur Verfügung gestellt werden (s. UML-Diagramm).



- Der für die Buttons zuständige ActionListener übernimmt lediglich die Steuerung, welche Methode aufgerufen wird und er reagiert auf die Rückgabewerte der bool'schen Methoden, indem er den Nachrichtentext entsprechend aktualisiert. Die einzelnen Buttons bekommen folgende Funktionen zugewiesen
  - Die Methode `ermittleObjekt()` der Klasse `Listenoperationen` durchsucht das `TreeSet` nach Einwohnerobjekten. Suchkriterien sind dabei der Name und Vorname des gesuchten Einwohners. Wird ein Objekt im `TreeSet` gefunden bei dem Name und Vorname mit den Suchkriterien übereinstimmen, wird das Objekt als Rückgabewert zurückgegeben, wird kein passendes Objekt gefunden, so gibt die Methode `null` zurück.
  - **Hinzufügen** – ruft die Methode `hinzufuegen()` der Klasse `Listenoperationen` auf, die wiederum auf Basis der übergebenen Parameter ein Einwohnerobjekt erzeugt und in das `TreeSet` einfügt. Der Erfolg oder Misserfolg des Einfügens soll im Nachrichtentext angezeigt werden (z.B. „Einwohner hinzugefügt“ oder „Einwohner existiert schon“)
  - **Anzeigen** – ruft die Methode `anzeigen()` der Klasse `Listenoperationen` auf. Die Methode ermittelt zunächst das anzuzeigende Objekt mithilfe der Methode `ermittleObjekt()`. Existiert ein entsprechendes Objekt im `TreeSet`, werden dessen Attribute auf der Konsole sowie ein entsprechender Nachrichtentext im Benutzerdialog ausgegeben. Wird kein passendes Objekt im `TreeSet` gefunden, so wird nur der Nachrichtentext im Benutzerdialog ausgegeben.
  - **Suchen** – ruft die Methode `suchen()` der Klasse `Listenoperationen` auf. Die Methode durchsucht das `TreeSet` mithilfe der Methode `ermittleObjekt()` und meldet im Nachrichtentext zurück, ob der Einwohner gefunden wurde oder nicht.
  - **Löschen** – ruft die Methode `loeschen()` der Klasse `Listenoperationen` auf. Mithilfe der Methode `ermittleObjekt()` wird zunächst geprüft, ob der zu löschende Einwohner im `TreeSet` vorhanden ist. Wird der zu löschende Einwohner im `TreeSet` gefunden, wird er aus dem `TreeSet` gelöscht. Der Erfolg bzw. Misserfolg des Löschens wird im Nachrichtentext des Benutzerdialogs angezeigt.
  - **Liste anzeigen** – ruft die Methode `listeAusgeben()` der Klasse `Listenoperationen` auf, die wiederum alle Objekte (z.B. in Form der wesentlichen Attribute) aus dem `TreeSet` auf der Konsole ausgibt.
- 4. Erweitern Sie Ihr Programm aus Übung 1 um ein geeignetes Menü (gleiche Funktionalitäten, wie die Buttons) und entsprechende Tooltips.