Inhaltsverzeichnis

1. Organisation & Nutzung von Java Klassen

Organisation

Die Ablage bzw. Speicherung eine Java Klassen in Form von Dateien erfolgt in Verzeichnissen, die ihrerseits hierarchisch angelegt werden (können aber nicht müssen). Im Kontext von Java werden diese Ablageverzeichnisse **Packages** genannt. Der Java Compiler registriert den Ablageort einer Java Klasse, indem in dieser ganz zu Beginn das Schlüsselwort

```
package
```

genutzt wird, zusammen mit der voll-qualifizierten Verzeichnis-Hierarchie unter Nutzung von Punkten als Trennzeichen, also etwa

```
package de.dhbw.domain.model;
```

Die *Eindeutigkeit* einer Java Klassen ergibt sich also nicht "allein" aus ihrem Namen der Klassensignatur, sondern nur in Verbindung mit dem package, in dem sie sich befindet. Beides ergibt den sog. **voll-qualifizierten Klassen-Namen**, etwa

```
de.dhbw.domain.model.Train
```

Importierung (Nutzung)

Wenn eine Java Klasse in/von einer anderen genutzt werden soll, muss sie zuerst **importiert** bzw. **bekannt gemacht** werden. Dazu muss sie mit ihrem *voll-qualifizierten Namen* (s.o.) ganz am Anfang der Datei angegeben werden. Das Package entspricht dabei dem Verzeichnis, in dem die Datei abgelegt ist. Dazu dient das **Schlüsselwort**

```
import
```

wie im folgenden Beispiel zu sehen:

```java
package de.dhbw.demo.model; ①

import java.util.List; ②

public class MyClass {
   // Inhalte der Klasse weggelassen ...
}
```

① Angabe des Speicherorts - des Verzeichnispfades - der Java Klasse

② Importierung von anderen Java Klassen, häufig aus separaten "Bibliotheken" (Dateien mit der Endung `.jar` → **J** ava **Ar** chive)

2. Klassen & Instanzen

[Inhalt | Demo | Übungen]

Kurz etwas **Theorie** vorab bzw. zur Wiederholung:

Unter einer **Klasse** (oder Objekttyp, →Ref. 1) versteht man in der **objektorientierten Programmierung** ein abstraktes Modell bzw. einen Bauplan für eine Reihe von ähnlichen Objekten.

Die Klasse dient als Bauplan für die Abbildung von realen Objekten in Softwareobjekte und beschreibt **Attribute** (auch Eigenschaften, Felder) und **Methoden** (bzw. Verhaltensweisen) der Objekte. Etwas allgemeiner kann auch gesagt werden, dass eine Klasse dem **Datentyp eines Objekts** entspricht.

Eine Java Klasse hat in der Regel folgende Bestandteile:

- Eigenschaft (Feld),
- Konstruktor und
- Verhaltensweise (Methode):

Die wichtigsten Elemente von Java:

Die **Klassensignatur** selbst:

```java
public class MyClass {

}
```

Eine Klasse mit einem **Feld** bzw. einer Eigenschaft:

```java
public class MyClass {

    public String field;
```

```
}
```

Eine Klasse mit (Default-) **Konstruktor** (engl. *constructor*, oder abgekürzt oft einfach `c'tor`):

In Java spielen die Konstruktoren eine wichtige Rolle bei der **Erzeugung von Objekten**. Sie sind spezielle Methoden, die aufgerufen werden, wenn ein neues Objekt erstellt wird, und ermöglichen es, das Objekt mit bestimmten Werten oder Eigenschaften zu initialisieren.

```
public class MyClass {

    public MyClass() {
        super(); // Aufruf des Default-C'tors der Elternklasse
    }
}
```

Eine Klasse mit einem **Konstruktor**, der ein Argument hat:

```
public class MyClass {

    public MyClass(String arg) {
        // Verarbeitung des Arguments,
        // z.B. mittels 'this' oder 'super'
    }

}
```

Die Erzeugung eines **Objektes** einer Klasse erfolgt durch die folgenden Schritte:

1. **Deklaration**: Die Variablendeklaration mit einem Variablennamen und einem Objekttyp
2. **Instanziierung**: Das Schlüsselwort `new` wird zum Erstellen des Objekts verwendet
3. **Initialisierung**: Auf das Schlüsselwort `new` folgt ein Aufruf eines Konstruktors. Dieser Aufruf initialisiert das neue Objekt.

```
               Instanzi-  Initiali-
Deklaration    ierung     sierung
|---------|     |-|       |------|

Train train  =   new       Train();
```

Eine Klasse mit einer **Methode** bzw. einer Verhaltensweise:

```
public class MyClass {

    public String doSomething() {
```

```
        return "result";
    }

}
```

Demo:

```
→ src/test/java/de/dhbw/demo/classes/ClassesDemoTest.java
```

Einschub: Zur **Theorie** des → Test Driven Developments, dann wieder hierher zurück!

Übungen:

```
→ src/test/java/de/dhbw/exercise/classes/ClassesExerciseTest.java
```

Erzeuge ein erstes, kleines **Klassenmodell**:

Übung 1

Folgendes soll implementiert werden:

- Eine Klasse `Fernzug` mit einem parameterlosen Konstruktor
- Eine Klasse `Regionalzug` mit einem parameterlosen Konstruktor
- Einen Test, der die korrekte Instanziierung der Instanzen bestätigt

Übung 2

- Ergänze die Klasse Zug mit einem Feld namens "number". Erzeuge eine Instanz und teste für die Instanz, welchen Wert dieses Feld einer Instanz hat!

Übung 3

Was fällt bei den Implementierungen besonders ins Auge?

3. Vererbung

[Inhalt | Demo | Übungen]

Bei der **Vererbung** in Java wird zwischen einer **Super**- und einer **Subklasse** unterschieden.

Die *Superklasse*, auch Eltern- oder Basisklasse genannt, ist vielfach eine Zusammenfassung von allgemeinen Attributen und Methoden unterschiedlicher aber ähnlicher Objekte (**Verallgemeinerung**).

Die *Subklasse*, auch Kind- oder Unterklasse bezeichnet, bekommt von ihrer Elternklasse sämtliche Attribute und Methoden vererbt, die nicht `private` sind. Des Weiteren kann die Subklasse um

eigene Attribute und Methoden erweitert werden. Man spricht hier von einer **Spezialisierung** der Subklasse von der Superklasse.

Eine Vererbung in Java findet über das Schlüsselwort

```
extends
```

statt. Der zugehörige Quellcode:

```
public class SuperClass { }

public class SubClass extends SuperClass { }
```

Die *grafische* Darstellung der Vererbung sieht folgendermaßen aus:

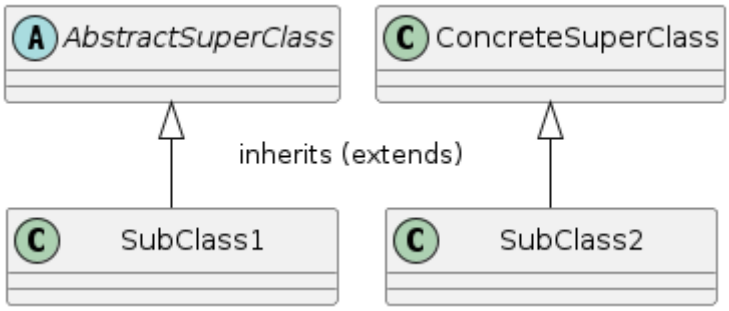


Figure 1. Inheritance

Einschub: "Der Objektvertrag" → ggf. Referenz auf `module-object-contract`

Demo:

```
→ src/test/java/de/dhbw/demo/inheritance/InheritanceTest.java
```

4. Polymorphismus

Die Methoden einer Klasse können der sogenannten **Polymorphie** unterliegen. Polymorphie ist griechisch und bedeutet *Vielgestaltigkeit*. Von Polymorphie spricht man in Java beispielsweise, wenn zwei Klassen denselben Methodennamen verwenden, aber die Implementierung der Methoden sich unterscheidet.

Häufig wird Polymorphie bei der **Vererbung** verwendet, d.h., dass einer Variablen nicht nur Objekte vom Typ der bei der Deklaration angegebenen Klasse zugewiesen werden können, sondern auch Objekte vom Typ der Kind-Klassen. Dies funktioniert nur, weil jede Kind-Klasse auch alle Methoden und Attribute ihrer Elternklassen implementieren muss.

Damit ist gewährleistet, dass alle Kind-Klassen über dieselben Methoden verfügen wie die

Elternklasse. Die Methoden können jedoch unterschiedlich implementiert werden, man spricht hier vom **"Überladen"** oder **"Übersteuern"** der Methode (→ mehr dazu siehe nachfolgender Abschnitt).

Demo:

```
→ src/test/java/de/dhbw/demo/inheritance/InheritanceDemoTest.java
```

In diesem Zusammenhang sind die Schlüsselwörter `super` und `this` von besonderer Bedeutung:

- `super`: Referenz für Felder oder Methoden der **Super-Klasse**
- `this`: Referenz für Felder oder Methoden aus der **eigenen Klasse**

Fragen:

1. *Warum kann Vererbung sinnvoll sein?*
2. *Wozu kann Polymorphismus nützlich sein?*

5. Überladen & Übersteuern (Überschreiben)

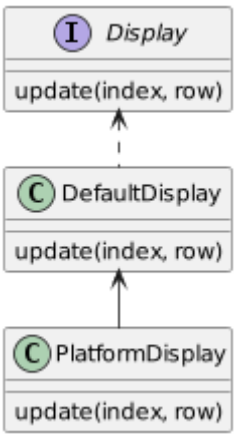
In einer Unterklasse können Methoden

- **überladen** (engl. "overload") oder
- **übersteuert** (engl. "override")

werden.

Von **Überladung** wird gesprochen, wenn eine Unterklasse neues Verhalten **hinzufügt**.

Das **Übersteuern** (synonym: Überschreiben) zeigt an, dass die Unterklasse geerbtes Verhalten **ersetzt**.



Beispiel Überladen von Methoden

```
public class PlatformDisplay {
```

```
    // multiple methods can be used to
    // update a platforms' display

    public void update() {}

    public void update(String line) {}

    public void update(List<String> lines) {}

}
```

Beispiel Übersteuern von Methoden

```
public interface Display {

    void update(int index, String row);
```

Die zugehörige **Annotation** (eine Art "Anzeiger") im Code ist

```
@Override
```

Sie sollte in jedem Fall genutzt werden, zudem wird sie auch von der IDE vorgeschlagen

Die zugehörigen **Unit-Tests** finden sich hier:

```
→ src/test/java/de/dhbw/demo/OverloadAndOverrideTest.java
```

Übungen:

Nutze, wann immer möglich, `Code Generierung` für die Schritte, am besten mithilfe der Tastenkombination.

▼ *Zur Erinnerung > Auszug aus 'Tools & Help...'*

Generierung von Quellcode (Klassen, Methoden, etc.)

1. mithilfe einer **Tastenkombination**
   a. unter MacOS: `Command + N`
   b. unter Windows: `Alt + Einfg` (ggf. auch `Fn + Alt + Einfg`) oder
2. mithilfe der **Maus**

Testklasse, in der die Übungen implementiert werden können:

```
→ src/test/java/de/dhbw/exercise/inheritance/InheritanceExerciseTest.java
```

Übung 1

Erstelle **Klassen** im Verzeichnis

```
src/main/java/de/dhbw/exercise/classes
```

1. Erstelle eine neue Klasse `Fernzug` mit einem parameterlosen Konstruktor
2. Erstelle eine neue Klasse `Regionalzug` mit einem parameterlosen Konstruktor
3. Teste, ob die Instanzen korrekt erzeugt wurden!

Übung 2

Übung zu **Syntax, Klassenmodell** und erforderliche **Methoden**.

- Eine Klasse `Zug` mit
  - einem String Feld `number` und
  - einer Methode `getNumber()`, die den Wert des gleichnamigen Feldes zurückgibt
- Eine Klasse `Fernzug`, die von der Klasse `Zug` erbt
- Ergänze die Klasse `Fernzug` mit einer beliebigen, deiner Meinung nach sinnvollen Spezialisierung
- Teste, welchen Wert das Feld `number` hat

Übung 3 *(optional)*

Nutze die Test-Klasse

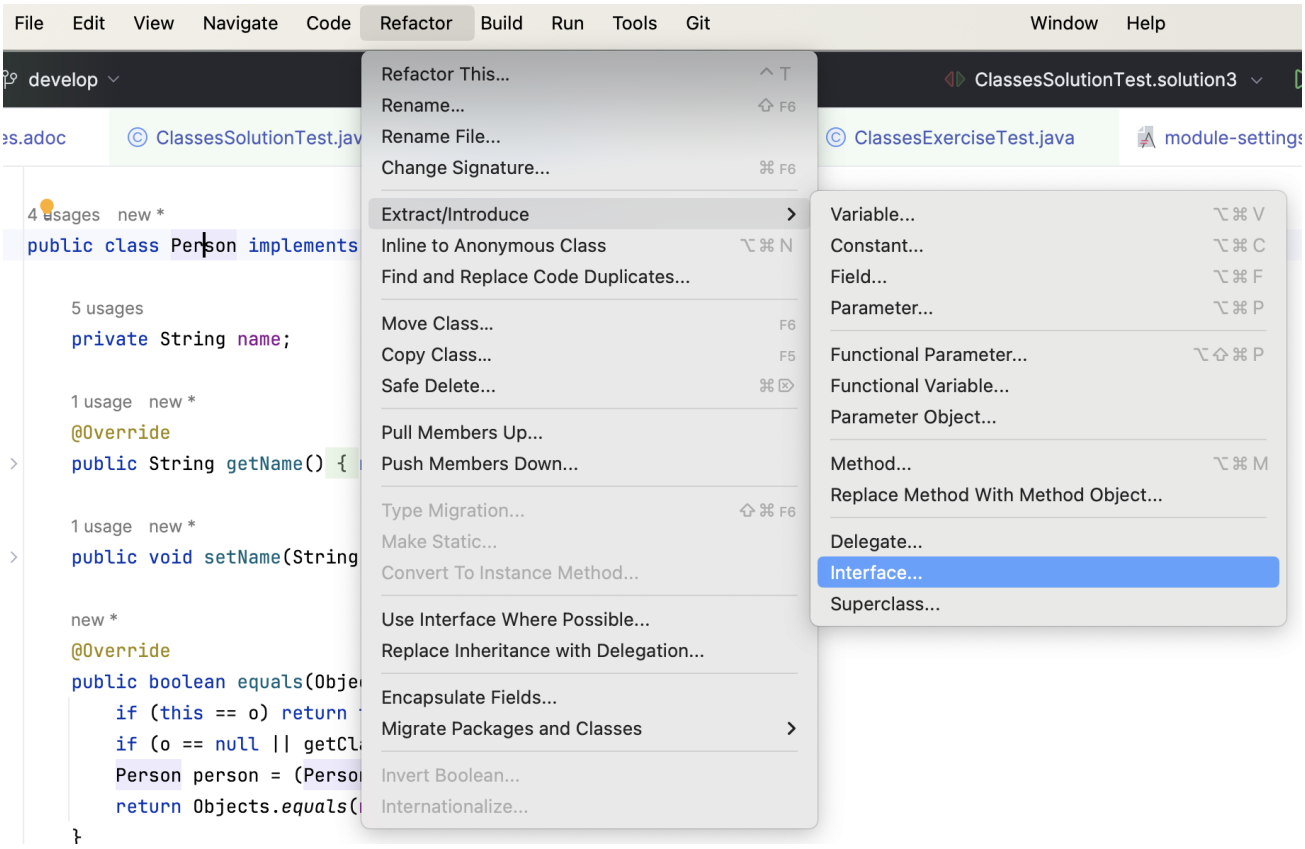
```
src/test/java/de/dhbw/exercise/classes/ClassesExerciseTest.java → exercise3()
```

Übung zu **Syntax, Klassenmodell** und erforderliche **Methoden**.

Erzeuge eine

1. konkrete Klasse `Person`
2. mit einem Attribut `name`, erstelle dann
3. die zum Attribut gehörende `get` und `set` Methode,
4. erstelle außerdem die `equals()` und `hashCode()` Methoden durch Code Generierung,
5. zuletzt leite aus dieser konkreten Klasse ein Interface `Mensch` ab

▼ *Hilfe zu 'Extract Interface ...'*



6. Referenzen

Ref. 1: https://de.wikipedia.org/wiki/Klasse_(Objektorientierung)