Klassen implementieren, Objekte instanziieren

Klasse implementieren

Jede neue Klasse implementiert man in ihrer eigenen Datei.
Die Datei muss zwingend wie die Klasse heissen.
Der Klassen-Name beginnt mit einem Grossbuchstaben.

Eine Klasse hat eine *Sichtbarkeit, einen Namen, Attribute, Konstruktoren und Getter/Setter sowie weitere Methoden*.

Hier ein Beispiel:

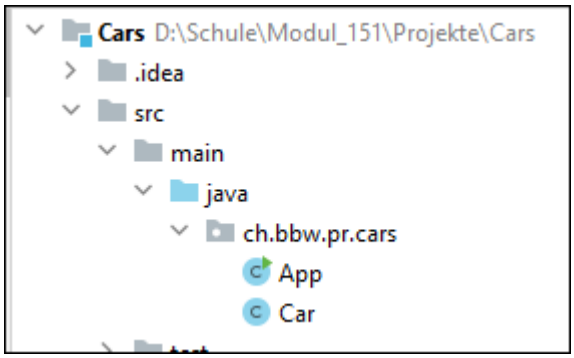
Car.java

```java
package ch.bbw.pr.cars;

/**
 * Car
 * Klasse für Autos
 *
 * @author Peter Rutschmann
 * @version 31.08.2021
 */
public class Car {
    //Attributes
    private String model;
    private double velocity;

    //Constructor without parameters
    public Car() {
        model = "unknown";
        velocity = 0;
    }

    //Constructor with parameters
    public Car(String model, double velocity) {
        this.model = model;
        this.velocity = velocity;
    }

    //Getter and Setter
    public String getModel() {
        return model;
    }

    public void setModel(String model) {
        this.model = model;
    }

    public double getVelocity() {
        return velocity;
    }

    public void setVelocity(double velocity) {
        this.velocity = velocity;
    }

    @Override
    public String toString() {
        return "Car{" + "model='" + model + '\'' + ", velocity=" + velocity + '}';
    }

    //other functions
    public void accelerate() {
        System.out.println("Car.accelerate(): is called, but no implementation available.");
    }
}
```

Die Reihenfolge *Attribute, Konstruktor, Getter/Setter und weitere Methoden* ist nicht zwingend, folgt aber einer allgemeinen Abmachung (Konvention).

Klassenkopf

```java
package ch.bbw.pr.cars;

/**
 * Car
 * Klasse für Autos
 *
 * @author Peter Rutschmann
 * @version 31.08.2021
 */
public class Car {
```



Das **Package** zeigt die Zugehörigkeit der Klasse zu einer bestimmten Komponente.
Das Package folgt der Konvention *<Land><Organisation><Unterorganisation><Komponente>*
(Im Beispiel ist die Unterorganisation mein Kürzel)

Der Java-Doc Kommentar informiert über die Klasse. Er hat zwei ** am Anfang.
Ein solcher Java-Doc Kommentar gehört zu einem guten Programmierstil.

Die Klasse beginnt mit der **Sichtbarkeit *public***, gefolgt vom Identifier ***class*** und dem gross geschriebenen **Klassennamen**.

Attribute

```java
public class Car {
    //Attributes
    private String model;
    private double velocity;
```

Die **Attribute**, *Merkmale* oder auch *Eigenschaften* erlauben es später dem instanziierten Objekt seinen Zustand speichern.

Ihre Sichtbarkeit ist meistens *private*.

Oft werden Attribute *english* geschrieben. Das ist zumeist *prägnanter* und *kompakter* als in Deutsch und *english* ist die Sprache der Programmierer.

Der Datentyp eines Attributes ist entweder:

ein Primitive Datatype:

https://docs.oracle.com/javase/tutorial/java/nutsandbolts/datatypes.html

zwei Beispiele

- int: Eine ganze Zahl
- double: Eine Gleitkomma Zahl für grössere Zahlen

Oder eine andere **Klasse**:

Beispiel

- String: eine Textzeile
- Color: eine Farbe (zBsp java.awt.Color)

Konstruktor

```java
//Constructor without parameters
public Car() {
    model = "unknown";
    velocity = 0;
}

//Constructor with parameters
public Car(String model, double velocity) {
    this.model = model;
    this.velocity = velocity;
}
```

Wenn von einer Klasse ein Objekt instanziiert wird, dann wird ein Konstruktor dieser Klasse aufgerufen.

Der Konstruktor ist eine Methode der Klasse.
Der Konstruktor erlaubt es, bei der Instanziierung den Zustand des Objektes zu setzen.
Also die Attribute mit Werten zu initialisieren.

Eine Klasse kann mehrere Konstruktoren haben.

Ein Konstruktor kann keine, ein oder mehrere Parameter haben.

Man muss sich immer überlegen, welcher Zustand das Objekt nach seiner «Konstruktion» haben soll. Also welche Werte die Attribute haben müssen.

Mit IntelliJ lassen sich Konstruktoren generieren: Menü *Code → Generate*

Getter/Setter

```java
//Getter and Setter
public String getModel() {
    return model;
}

public void setModel(String model) {
    this.model = model;
}

public double getVelocity() {
    return velocity;
}

public void setVelocity(double velocity) {
    this.velocity = velocity;
}
```

Der **Zugriff von aussen auf ein Attribut** sollte nie direkt auf das Attribut geschehen.
Für den Zugriff verwendet man die **Getter und Setter-Methoden**.

this. Steht für das aktuelle Objekt. zBsp: this.model = model;
Dem Attribut model des aktuellen Objekts wird der Wert des Parameters model zuweisen

Mit IntelliJ generieren über: Menü *Code → Generate*

Frameworks verwenden/verlangen oft die Getter/Setter Methoden.

Weitere Methoden

```
44      @Override
45      public String toString() {
46          return "Car{" + "model='" + model + '\'' + ", velocity=" + velocity + '}';
47      }
```

Die **toString Methode** ist eine spezielle Methode. Sie gibt den **Zustand des Objektes** als String zurück. Also zumeist die Werte aller Attribute.
Dies dient der Ausgabe, dem Logging oder kann das Debugging unterstützen.

Mit IntelliJ generieren über: Menü *Code → Generate*

Und dann gibt es noch weitere Methoden, die das **Verhalten des Objektes** definieren.

Die gezeigte Methode ist vom Inhalt her eher einfach.

Die effektive Implementierung fehlt noch.

```
49      //other functions
50      public void accelerate() {
51          System.out.println("Car.accelerate(): is called, but no implementation available.");
52      }
```

Objekt instanziieren

Objekte einer Klasse werden in anderen Klassen in deren Konstruktoren oder Methoden instanziiert.

App.java

```
1  package ch.bbw.pr.cars;
2
3  import java.util.ArrayList;
4  import java.util.List;
5
6  /**
7   * Application
8   * Hier startet das Programm
9   * @author Peter Rutschmann
10  * @date 31.08.2021
11  */
12 public class App {
13     public static void main(String[] args) {
14         System.out.println("Cars Application");
15
16         Car myFirstCar = new Car();
17         myFirstCar.setModel("Ford");
18         System.out.println(myFirstCar);
19         System.out.println(myFirstCar.getModel());
20         myFirstCar.accelerate();
21
22         Car mySecondCar = new Car( model: "Opel",  velocity: 100.0);
23
```

A)

B)

Output auf der Console:

```
Cars Application
Car{model='Ford', velocity=0.0}
Ford
Car.accelerate(): is called, but no implementation available.
```

A) Sie ***deklarieren*** eine Variable vom Typ *Car* und **instanziieren** das Objekt mit ***new*** und dem Aufruf des ***Konstruktors***.
Sie weisen der Variablen eine Referenz auf das Objekt zu.

B) Sie ***deklarieren*** eine Variable vom Typ *Car* und **instanziieren** das Objekt mit ***new*** und dem Aufruf des ***Konstruktors*** mit Parametern und ***initialisieren*** so das Objekt.
Sie geben direkt das model und die velocity mit.
Sie weisen der Variablen eine Referenz auf das Objekt zu.

Anschliessend können Sie den Zustand mit einem Setter neu setzen oder einem Getter abfragen.

Oder mit Methoden wie accelerate das Verhalten des Objektes aufrufen.

Beim Aufruf von System.out.println(myFirstCar) wird der Zustand des Objekt myFirstCar als Text ausgegeben. Java erkennt, dass es dazu die toString Methode der Klasse verwenden muss, ohne dass ich die Methode selber aufrufen muss.

Aufgabe

```
24        //geht nicht, doch wieso?
25        System.out.println(myFirstCar.model);
```

Dieser Aufruf funktioniert nicht

Doch was ist der Grund dafür?[1]

Bitte beachten:
Es gibt Frameworks, die erkennen, dass sie automatisch den Getter an Stelle von model verwenden müssen.

Mehrere Objekte in einer Liste verwalten

Die Container-Klassen wie *List* vereinfachen es, mehrere Objekte der gleichen Klassen zu verwalten.

```
27        //multiple cars in a list of Car
28        List<Car> myCars = new ArrayList<>();
29        myCars.add(myFirstCar);
30        myCars.add(mySecondCar);
31        System.out.println(myCars);
```

```
[Car{model='Ford', velocity=0.0}, Car{model='Opel', velocity=100.0}]
```

Mehrere Objekte in einer Liste verwalten und direkt instanziieren und initialisieren

Müssen von einer Klasse mehrere Objekte instanziiert, initialisiert und in Liste verwaltet werden, so lässt sich das in einer sehr kompakten Art und Weise machen.

```
33        //multiple cars in a list of cars, direct initialized
34        List<Car> myOtherCars = new ArrayList<>();
35        myOtherCars.add(new Car( model: "Volvo",  velocity: 50.0));
36        myOtherCars.add(new Car( model: "Renault",  velocity: 80.0));
37        System.out.println(myOtherCars);
```

```
[Car{model='Volvo', velocity=50.0}, Car{model='Renault', velocity=80.0}]
```

[1] Das Attribut *model* im der Klasse *Car* ist *private* deklariert und damit kann nicht darauf zugegriffen werden. Wir müssen die Methode getModel() verwenden.