# Objektorientierte Programmierung

UML und Interfaces

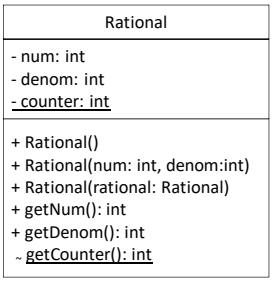
Prof. Dr. Ulrike Hammerschall
Fakultät für Informatik und Mathematik

## UML Klassendiagramme

- Die Unified Modeling Language (UML) ist eine grafische Sprache, die u.a. zur Darstellung von Programmstruktur und Programmablauf (Kontrollfluss) von Softwaresystemen verwendet werden kann.
- UML Klassendiagramme sind eine Notation der UML. Sie werden verwendet um die Programmstruktur im Detail darzustellen.
- Ein Klassendiagramm stellt graphisch dar,
  - aus welchen Klassen und Interfaces sich das Programm zusammensetzt,
  - was deren wichtigste Elemente (Konstruktoren, Methoden, Objektvariablen, Klassenvariablen, Konstanten) sind,
  - in welchen Beziehungen die Objekte der Klassen zueinander und zu den Interfaces stehen.
- Ein Klassendiagramm liefert die *Landkarte* eines Softwareprogramms: schneller Überblick, leichtere Orientierung.

## Darstellung einer Klasse mit UML

- Drei Bereiche
  - Name der Klasse
  - Objektvariablen, Klassenvariablen, Konstanten
  - Konstruktoren und Methoden
- Modifier
  - – für private-Elemente
  - + für öffentliche Elemente
  - ~ für Package-Sichtbarkeit
- Klassenvariablen / -methoden
  - Unterstrichen
- Mit Ausnahme des Namens sind alle Elemente optional.

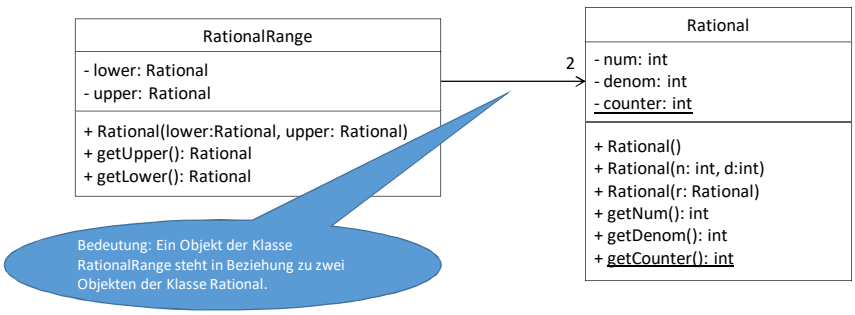



*Beispiel Klasse Rational als UML Klasse*

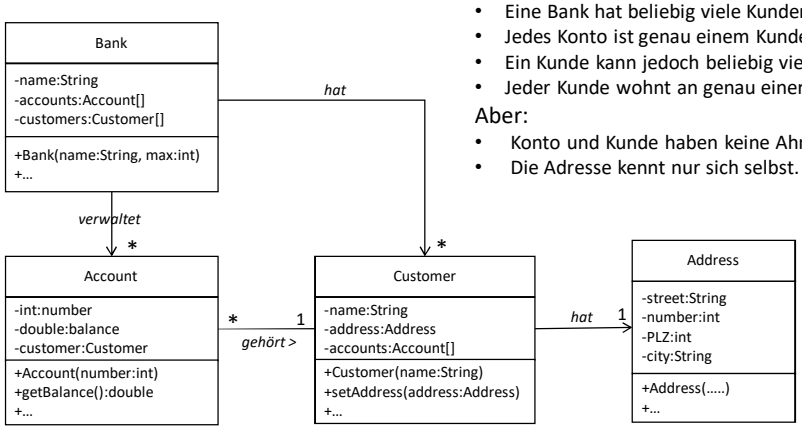


## Beziehungen

- Beziehungen zwischen Klassen werden über Assoziationen ausgedrückt.
- Die Richtung des Pfeils gibt die Richtung der Beziehung an. Keine Pfeilspitzen entsprechen einem Pfeil in beide Richtungen.
- Die Multiplizität gibt an, wie viele Objekte zu dieser Klasse an einer solchen Beziehung beteiligt sind (z.B. 0, 1, 0..1, 1..*, *, konkrete Zahl).



## Klassendiagramm am Beispiel Bank




Ein Klassendiagramm beschreibt eine Geschichte zum Programm:

- Eine Bank hat beliebig viele Konten (Account).
- Eine Bank hat beliebig viele Kunden (Customer).
- Jedes Konto ist genau einem Kunden zugeordnet.
- Ein Kunde kann jedoch beliebig viele Konten haben.
- Jeder Kunde wohnt an genau einer Adresse.

Aber:

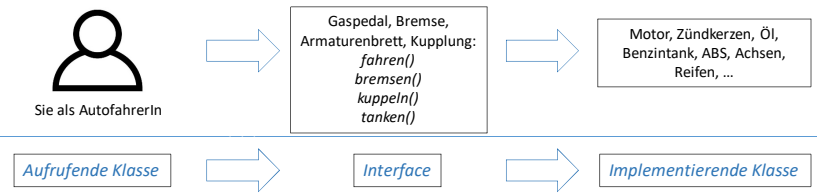
- Konto und Kunde haben keine Ahnung von der Bank.
- Die Adresse kennt nur sich selbst.



## Interfaces in Java

- **Interfaces** (Schnittstellen) sind neben Klassen ein weiteres Element objektorientierter Programme.
- Ein Interface definiert einen **Vertrag** zwischen einer **Klasse** und dem **Aufrufer** dieser Klasse.
- Als Vertrag definiert das Interface die **Signaturen** der öffentlichen (**public**) Methoden, die die Klasse implementieren muss.
- Das Interface legt dabei **nicht** fest, wie die Methoden zu implementieren sind.

*Eine kleine Analogie:*



## Implementierung von Interfaces

- Interfaces sind für sich genommen nicht anwendbar. Es braucht immer eine Implementierung.
- Konkrete Klassen implementieren das Interface, d.h. definieren die vollständigen Methoden zu den Signaturen im Interface.
- Der Compiler prüft, ob zu allen Signaturen im Interface eine entsprechende Implementierung in der Klasse existiert.



## Definition der Schnittstelle in Java – Beispiel Printable

- Das Interface (Schnittstelle) ...

```
public interface Printable {
        void print();
}
```

- fordert, dass jede Klasse, die diese Schnittstelle implementiert, eine öffentliche Implementierung der Methode print liefern muss:

```
public void print() {
        …
}
```

## Anwendung der Schnittstelle – Klasse Book

- Eine Klasse, welche die Schnittstelle implementiert, kennzeichnet dies über das Schlüsselwort: *implements*

```
public class Book implements Printable {
        public void print() {
            System.out.println("Druck auf Laserdrucker");
        }
        // Beliebige weitere Methoden der Klasse Book
        …
}
```



## Anwendung der Schnittstelle – Klasse Poster

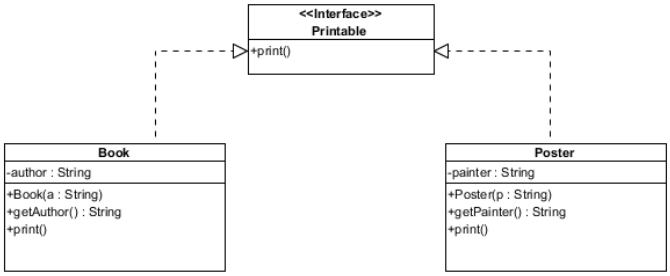
- Das Interface kann von beliebig vielen Klassen auf unterschiedliche Art implementiert werden.

```
public class Poster implements Printable {
        public void print() {
            System.out.println("Druck auf Plotter");
        }

        // Beliebige weitere Methoden der Klasse Poster
        …
}
```

## Darstellung mit der UML

- Interfaces erhalten das Stereotype *<<interface>>*.
- Sie haben i.d.R. nur einen Bereich für den Namen und einen für die Methoden.
- Darstellung der Beziehung *implements* in der UML über einen gestrichelten, nicht ausgefüllten Pfeil.






## Erweiterungen seit Java 8

- Seit Java 8 zwei weitere Arten von Methoden in Interfaces:
  - Default-Methoden:
    - Schlüsselwort **default**. Methoden mit fertiger Implementierung.
    - Stehen allen implementierenden Klassen zur Verfügung.
    - Zugriff über Objekte. dürfen jedoch keine konkreten Objektvariablen nutzen.
    - Nur Methoden der Schnittstelle sind verfügbar.
  - Statische Methoden:
    - Schlüsselwort **static**. Methoden gehören dem Interface. Zugriff über Interface-Namen.
- Hintergrund: Umfangreiche Erweiterungen der Sprache machten diese Änderung notwendig um bestehende Interfaces in der Java-Bibliothek erweitern zu können ohne bestehende Implementierungen der Interfaces zu belasten.

## Default- und statische Methoden - Interface

```
public interface MyInterface {
        int VALUE = 100;          // Konstante (public static final implizit festgelegt)

        int getFirst();           // Abstrakte Methode
        int getSecond();          // Abstrakte Methode

        // default Methode
        default int getSum() {
                return getFirst() + getSecond();
        }

        // statische Methode
        static int getBoundary() {
                return VALUE * VALUE;
        }
}
```



## Default- und statische Methoden – implementierende Klasse

```
public class MyClass implements MyInterface {
        private final int first;
        private final int second;
        MyClass(final int first, final int second) {
                this.first = first;
                this.second = second;
        }
        @Override
        public int getFirst() {   // Implementiert die abstrakte Methode getFirst im Interface
                return first;
        }
        @Override
        public int getSecond() {  // Implementiert die abstrakte Methode getSecond im Interface
                return second;
        }
}
```

## Default- und statische Methoden – Anwendung

```
public class MyMain {

    public static void main(final String[] args) {

        MyInterface c = new MyClass(2, 3);

        c.getFirst();     // Methode der Klasse selbst
        c.getSecond();    // Methode der Klasse selbst
        c.getSum();       // Aufruf der Default-Methode aus Interface

        // Aufruf der statischen Methode aus dem Interface.
        // Weder auf Objekten, noch in der implementierender Klasse verfügbar.
        MyInterface.getBoundary();
    }
}
```