# Objektorientierte Programmierung

Objektorientierung aus der Vogelperspektive

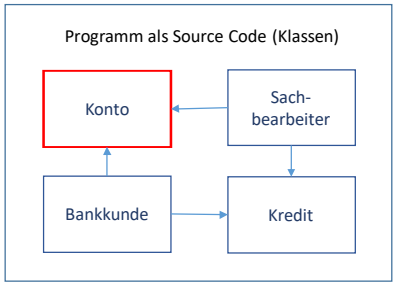
Prof. Dr. Ulrike Hammerschall
Fakultät für Informatik und Mathematik

## The Big Picture

- Der Quellcode eines Java-Programms besteht aus einer beliebigen Menge von **Klassen**.
- Zur Laufzeit erzeugt der Interpreter anhand der Klassen konkrete **Objekte**.
- Die Objekte rufen gegenseitig ihre **Methoden** auf und simulieren so das gewünschte Verhalten.
- Die Methoden verwenden **Objektvariablen** um berechnete Werte für einzelne Objekte während des Programmablaufs zu speichern.
- Diese Elemente sind die wesentlichen (aber noch nicht alle) Ingredienzien eines Java-Programms.

## Klassen und Objekte am Beispiel Bank


Reale Welt: Eine Bank hat Kunden, die ihr Geld auf Konten verwalten wollen und hin und wieder Kredite benötigen. Sachbearbeiter unterstützen die Kunden am Schalter.






## Ein Beispiel: Die Klasse Account (Konto)

```
class Account {

    // Kontonummer (number)
    // Kontostand (balance)

    // einzahlen (deposit)
    // abfragen (getBalance)

}
```

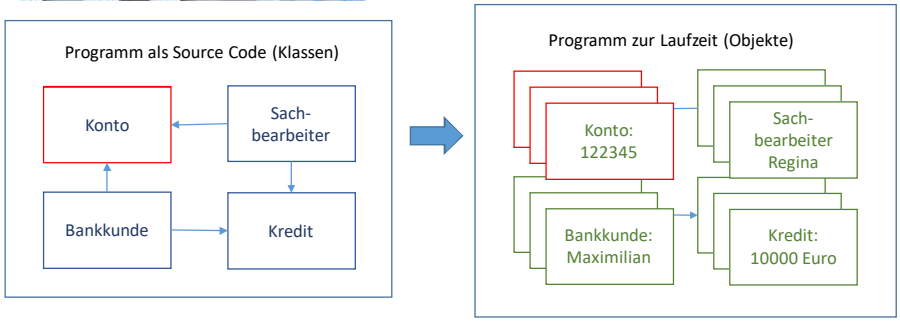
Rahmendefinition einer Klasse

Objektvariablen, beschreiben die Eigenschaften der Objekte zur Klasse

Methoden, beschreiben das Verhalten der Objekte zur Klasse

## Klassen und Objekte am Beispiel Bank


Realen Welt: Eine Bank hat Kunden, die ihr Geld auf Konten verwalten wollen und hin und wieder Kredite benötigen. Sachbearbeiter unterstützen die Kunden am Schalter.






## Objektvariablen der Klasse Account

**Klasse**

```
class Account {

    // Kontonummer
    private int number;
    // aktueller Kontostand
    private double balance = 0;

    // einzahlen (deposit)
    // abfragen (getBalance)

}
```

**Objektvariablen**

Die Definition von **Objektvariablen** erfolgt:
- **immer** innerhalb der Klasse und
- **niemals** innerhalb einer Methode.

**Methoden**

## Methoden der Klasse Account

Klasse

```
class Account {
    ...                                          Objektvariablen
    // Einzahlungen in das Konto
    void deposit(double amount) {
        balance += amount;
    }
    // Abfrage des aktuellen Kontostands
    double getBalance() {
        return balance;
    }                                            Methoden
}
```

die Definition von Methoden erfolgt:
- **immer** innerhalb der Klasse (nach den Objektvariablen),
- **niemals** innerhalb einer anderen Methode.



## Objektvariablen und Methoden

### Eine Objektvariable:

- hat einen **Datentyp** (Basistyp oder Referenztyp),
- hat einen eindeutigen **Namen** (Identifier),
- erhält im Laufe des Programmablaufs **Werte** zugewiesen.
- hat eine **Sichtbarkeit** (visibility): normalerweise private.

### Eine Methode

- kapselt einen **Algorithmus**,
- hat einen eindeutigen **Namen**,
- erhält ggf. bestimmte Werte (**Argumente**) als Eingabe,
- berechnet ein **Ergebnis** und gibt es zurück,
- definiert und nutzt eigene **lokale Variablen**,
- hat Zugriff auf alle **Objektvariablen** der Klasse.

## Aufbau von Methoden

```
<Ergebnistyp> <Methodenname> (<Parameterliste>) {
```

Methodenkopf

```
      <beliebig lange Liste von Anweisungen>
      <ggf. Rückgabe des Ergebnisses>
```

Methodenrumpf

```
}
```

Zum Vergleich…

```
// Einzahlungen in das Konto
void deposit(double amount) {
      balance = balance + amount;
}
```



## Definition von Methoden – Ergebnistyp

- Datentyp des berechneten Ergebnisses.

```
// Einzahlungen in das Konto
void deposit(double amount) {
      balance = balance + amount;
}
// Abfrage des aktuellen Kontostands
double getBalance() {
      return balance;
}
```

- Spezieller Datentyp **void**: kennzeichnet, dass kein Ergebnis von der Methode erwartet wird. Ist nur an dieser Stelle erlaubt!

## Definition von Methoden – Methodenname

- Innerhalb der Klasse eindeutiger Bezeichner für die Methode.
- Methodennamen beschreiben normalerweise eine aktive Handlung.
- Es gelten die gleichen Namenskonventionen wie für Variablen.

```
// Einzahlungen in das Konto
void deposit(double amount) {
	balance = balance + amount;
}
// Abfrage des aktuellen Kontostands
double getBalance() {
	return balance;
}
```



## Definition von Methoden – Parameterliste

- Beliebig lange Liste von Parametern, darf auch leer sein.
- Mehrere Parameter werden mit Komma getrennt.
- Parameter werden wie Variablen definiert.
- Das Schlüsselwort final verhindert, dass der Parameter im Methodenrumpf verändert wird.

```
// Einzahlungen in das Konto
void deposit(double amount) {
	balance = balance + amount;
}
// Abfrage des aktuellen Kontostands
double getBalance() {
	return balance;
}
```

## Definition von Methoden - Methodenrumpf

- Führt die Berechnungen durch. Besteht aus beliebigen Listen von Anweisungen.
- die Rückgabe von berechneten Werten erfolgt mit **return**. Der Wert muss dem Ergebnistyp der Methode entsprechen. Methoden mit Rückgabetyp void benötigen kein return.

```
// Einzahlungen in das Konto
void deposit(double amount) {
        balance = balance + amount;
        return;
}
// Abfrage des aktuellen Kontostands
double getBalance() {
        return balance;
}
```



## Erzeugen von Objekten

- Der Operator zum Erzeugen neuer Objekte ist **new**
- Das erzeugte Objekt kann einer Variable vom Typ des Objekts (entspricht dem Klassennamen) zugewiesen werden.
- Struktur der Instanziierung:

```
<Datentyp> <Variablenname> = new <Datentyp>();
```

- Beispiel:

```
Account myAccount = new Account();
```

## Aufruf einer Methode

HM

- Innerhalb des gleichen Objekts über Methodennamen

```
double value = getBalance();
```

- auf fremden Objekten über Punkt-Operator

```
Account myAccount = new Account();
double value = myAccount.getBalance();
```

- Ergebnis kann einer Variable vom Ergebnistyp der Methode zugewiesen werden.
- Achtung: Der Aufruf einer Methode der Klasse aus der main-Methode heraus entspricht dem Aufruf auf einem fremden Objekt.



## Zugriff auf Objektvariablen

HM

- Aufruf von Objektvariablen
  - im gleichen Objekt über den Variablennamen.
  - auf fremden Objekten über Punkt-Operator. Nur möglich bei Objektvariablen, die nicht private deklariert sind.
- Sollte aus Gründen der Datenkapselung vermieden werden.
- Alternative: Getter- und Setter-Methoden.

Problem: Wann werden Objektvariablen Initialisiert bzw. bekommen einen initialen Wert?

## Variante 1: Objektvariable erhält direkt Initialwert

```
class Account {
        // Kontonummer
        private int number;
        // aktueller Kontostand
        private double balance = 0;

        // Einzahlungen in das Konto
        void deposit(double amount) {
                balance = balance + amount;
        }
        // Abfrage des aktuellen Kontostands
        double getBalance() {
                return balance;
        }
}
```

- Ideal, wenn der Wert zu Beginn für alle Objekte gleich sein soll.
- **Aber**, viele Werte kennzeichnen die Individualität eines Objekts.
- Nur möglich wenn für alle Objekte der gleiche (Start-)Wert sinnvoll ist.



## Variante II: Explizites Setzen über Methoden

```
class Account {
        // Kontonummer
        private int number;
        // aktueller Kontostand
        private double balance = 0;

        // Setze Kontonummer
        void setNumber(int nr) {
                number = nr;
        }
        // Liefere Kontonummer
        int getNumber() {
                return number;
        }
        …
```

- Ok, wenn die Werte bei der Erstellung der Objekte nicht bekannt sind und erst später gesetzt werden sollen.
- **Aber**: es kann die Situation eintreten, dass ein Objekt in einem inkonsistenten Zustand verwendet wird.

## Variante III: Setzen des Werts im Konstruktor

- Spezielle Methode, die der Klasse selbst zugeordnet ist.
- Liefert als Ergebnis ein Objekt der Klasse.
- Erlaubt die Zuordnung von Werten zu Objektvariablen bei der Initialisierung.
- Syntax:

```
<Klassenname>(<Parameterliste>) {
   // Initialisierung der Objektvariablen
}
```



## Default-Konstruktor

- Konstruktor mit leerer Parameterliste.
- Es werden keine Werte von außen übergeben.
- Es können jedoch beliebige Berechnungen oder Wertzuweisungen vorgenommen werden.

```
// Kontonummer
private int number;

// Default-Konstruktor
Account() {
       number = 123498765;
}
```

```
// Initialisierung eines Objekts
Account myAccount = new Account();
```

## Custom-Konstruktor

- Konstruktor mit nicht-leerer Parameterliste.
- Der Konstruktor erwartet konkrete Werte über seine Parameterliste.

```
// Kontonummer
private int number;

// Custom-Konstruktor
Account(int nr) {
        number = nr;
}
```

```
// Initialisierung eines Objekts
Account myAccount = new Account(12349876);
```



## Generierter Default-Konstruktor

- Es wird kein Konstruktor in der Klasse definiert.
- Der Compiler erzeugt in diesem Fall automatisch einen leeren Konstruktor und ruft diesen auf.

```
class Account {
        …
}
```

```
// Initialisierung eines Objekts
Account myAccount = new Account();
```

## Zusammenfassung

- Objektorientierte Programme fassen Eigenschaften und Verhalten ähnlicher Objekte der Realität zu Klassen zusammen.
- Eigenschaften werden über **Objektvariablen** repräsentiert, Verhalten über **Methoden**.
- Objekte zu einer Klasse werden über den Aufruf eines Konstruktors der Klasse erzeugt. Konstruktoren sind ähnlich zu Methoden aufgebaut und werden ebenfalls innerhalb der Klasse definiert. Der Operator für den Konstruktoraufruf ist **new.**
- Jede Klasse hat automatisch mindestens einen **Konstruktor** (auch wenn er nicht explizit definiert wurde).
- Im Konstruktor (also bei der Initialisierung eines Objekts) erhalten die Objektvariablen initiale Werte. Diese können im Programmablauf über Methodenaufrufe verändert werden.



## Übung in Gruppen

- Entwerfen Sie gemeinsam (auf Papier) eine Klasse für einen Kaffeeautomaten.
- Der Automat hat einen Wassertank mit 500 ml Fassungsvermögen und einen Behälter für Kaffeepulver mit 100 g Fassungsvermögen.
- Zum Kochen einer Tasse Kaffee benötigt der Automat sowohl Wasser (200 ml) als auch Pulver (20 g).
- Man kann eine Tasse Kaffee kochen. Falls Kaffee oder Wasser nicht mehr ausreichen, muss manuell nachgefüllt werden.
- Überlegen Sie welche Eigenschaften der Kaffeeautomat haben sollte. Welche Datentypen benötigen Sie?
- Welches Verhalten hat der Kaffeeautomat. Definieren Sie nur die Methodensignaturen.
- Hausaufgabe: Implementieren Sie den Kaffeeautomaten.