Wie definiert man eine Klasse?

public class *NameDerKlasse* {

Instanzvariablendeklarationen

Methodendefinitionen

Konstruktordefinitionen }

Die drei Komponenten einer Klassendefinition können in beliebiger Reihenfolge und auch vermischt angegeben werden.

Beispiel Bankkonto

```
public class Bankkonto {
    Bankkonto() { kontostand = 0; }

    public void einzahlen(double betrag)
       { kontostand = kontostand + betrag;}

    private double kontostand;

    public void abheben(double betrag)
       { kontostand = kontostand - betrag; }

    public double getKontostand()
       { return kontostand; }
}
```


Beispiel Bankkonto (besser)

```
public class Bankkonto {
    private double kontostand;

    Bankkonto() { kontostand = 0; }

    public void einzahlen(double betrag)
       { kontostand = kontostand + betrag;}

    public void abheben(double betrag)
       { kontostand = kontostand - betrag; }

    public double getKontostand()
       { return kontostand; }
}
```


Deklaration einer Instanzvariablen

Die Deklaration einer Instanzvariablen hat die Form

`private` *typ variablenName* `;`

Hier ist *variablenName* der Name der deklarierten Variablen. und *typ* ist der Typ der Instanzvariablen.

Der Typ kann entweder ein Basistyp sein

`double, int, char, boolean, long, float`

...oder eine vordefinierte Klasse

`String, Point, Rectangle, Random, ...`

...oder eine selbstdefinierte Klasse

`Bankkonto, Dreieck, Betrunkener, ...`

.

Bedeutung der Typen

Das bedeutet, dass die Instanzvariable nur Werte dieses Typs enthalten darf.

Im Gegenzug ist es möglich, die Instanzvariable mit Operationen dieses Typs zu verarbeiten.

Ist etwa `x` vom Typ `String`, dann darf man `x.length()` schreiben.

Definition einer Methode

`public` *ergebnisTyp methodenName* (typ_1 par_1 , typ_2 par_2 , ... , typ_n par_n) {

methodenRumpf

}

- *methodenName* ist der Name der Methode. Ist e ein Objekt der Klasse, dann ruft man die Methode so auf: e.*methodenName*(...)
- Die Variablen $par_1, \ldots, par_n$ sind die Parameter. Beim Methodenaufruf muss man konkrete Werte für die Parameter bereitstellen.
- Die Typen $typ_1, \ldots, typ_n$ sind die Typen der Parameter (Basistypen oder Klassen).

  Beim Methodenaufruf müssen die konkreten Parameter den angekündigten Typ haben.

Definition einer Methode

- Der Typ *ergebnisTyp* ist der Typ des Ergebnisses des Methodenaufrufs. Ist er `void`, so wird kein Ergebnis zurückgegeben.

Beispiele aus `Bankkonto`

```
public void einzahlen(double betrag) {
    ...
}
public boolean equals(Bankkonto konto) {
    ...
}

public void ueberweisen(double betrag, Bankkonto empfaenger) {
    ...
}

public double getKontostand() {
    ...
}
```

Der Methodenrumpf

- Der Methodenrumpf kommt zur Ausführung, wenn die Methode bei einem Objekt der Klasse aufgerufen wird.
- Im Methodenrumpf sind die Instanzvariablen, sowie die Parameter verfügbar. Außerdem möglicherweise deklarierte lokale Variablen.
- Der aktuelle Wert der Instanzvariablen ist bestimmt durch das Objekt, bei dem die Methode aufgerufen wird.
- Der aktuelle Wert der Parameter ergibt sich aus den konkreten Parametern, mit denen die Methode aufgerufen wird.
- Man kann sagen, dass die Instanzvariablen Teile eines *impliziten Parameters* sind.
- Hat die Methode einen von `void` verschiedenen Rückgabewert, so muss der Rumpf ein entsprechendes `return`-Statement enthalten.

Das `return`-Statement

Das `return`-Statement hat die Form:

return e;

wobei e ein Ausdruck ist.

Der Ausdruck e muss als Typ den Ergebnistyp der Methode, in der das `return` Statement vorkommt, haben.

Gelangt der Kontrollfluss (*vulgo* "der Computer") an das Statement

return e;

so wird e ausgewertet und die Methodenabarbeitung beendet. Rückgabewert der Methode ist dann der Wert von e.

Beispiele

```
public void einzahlen(double betrag) {
    kontostand = kontostand + betrag;
}

public void abheben(double betrag) {
    kontostand = kontostand - betrag;
}

public boolean equals(Bankkonto konto) {
    return kontostand == konto.getKontostand();
}

public double getKontostand() {
    return kontostand;
}
```


Überweisen

```
public void ueberweisen(double betrag, Bankkonto empfaenger) {
    kontostand = kontostand - betrag;
    empfaenger.einzahlen(betrag);
    }
```

auch richtig, evtl. sogar besser:

```
public void ueberweisen(double betrag, Bankkonto empfaenger) {
    this.abheben(betrag);
    empfaenger.einzahlen(betrag);
}
```

Der Ausdruck `this` bezeichnet das Objekt, bei dem die Methode aufgerufen wird.

Man darf `this` auch weglassen:

```
abheben(betrag);
```

`this` bei Instanzvariablen

Statt `kontostand` kann man auch `this.kontostand` schreiben.

Das ist nützlich, wenn man eine lokale Variable desselben Namens wie eine Instanzvariable deklariert hat.

Letztere würde nämlich die Instanzvariable unsichtbar machen.

```
public double getKontostand() {
    String kontostand = "Abrakadabra";
    return this.kontostand;
}
```

Wer will, kann immer `this` schreiben, um auf Instanzvariablen zuzugreifen.

Seiteneffekte

Verändert eine Methode die Instanzvariablen von anderen Objekten als `this` so liegt ein Seiteneffekt vor.

Beispiel: die Methode `ueberweisen` verändert die Instanzvariable des zweiten Parameters.

Seiteneffekte sollten so wenig wie möglich verwendet werden und auf jeden Fall nur da, wo wirklich sinnvoll.

Tastatureingaben und Bildschirmausgaben sind auch Seiteneffekte. Auch deren Verwendung innerhalb von Methoden sollte minimiert werden.

Schlechtes Beispiel: `getKontostand()` so implementiert, dass gleichzeitig auf den Bildschirm geschrieben wird.

Beispiel: Verzinsung

```
public double zinsenVorhersagen(int jahre, double zinssatz) {
   double stand = this.kontostand;

   for (int i = 1; i <= jahre; i++) {
      stand = stand + zinssatz / 100. * stand;
   }
   return stand;
}
```


Drei Arten von Variablen

Es gibt drei Arten von Variablen:

- Instanzvariablen (z.B. `kontostand`)
- Parameter (z.B. `betrag`)
- Lokale Variablen (z.B. `stand`, `i`)

Jede Variable wird irgendwann mit ihrem Typ deklariert.

Jede Variable hat eine Lebensdauer, während der sie im Speicher repräsentiert ist. Während der Lebensdauer kann auf ihren Wert zugegriffen werden und ihr Wert verändert werden (durch `=`).

Jede Variable hat auch einen Sichtbarkeitsbereich im Programm. Man darf sie nur innerhalb ihres Sichtbarkeitsbereichs verwenden.

Die Sichtbarkeitsbereiche stellen sicher, dass beim Ablauf des Programms nie versucht wird, auf eine Variable zuzugreifen, deren Lebensdauer bereits um ist.

Instanzvariablen

- *Instanzvariablen* werden in einer Klasse deklariert; mit jedem (durch `new` erzeugten) Objekt der Klasse wird ein neuer Satz dieser Instanzvariablen ins Leben gerufen.
- Sie werden gemäß dem Konstruktor oder “per default” initialisiert.
- Die Lebensdauer einer Instanzvariablen ist gleich der Lebensdauer des Objekts, welches sie enthält.
- Eine Instanzvariable ist sichtbar in allen Methodenrümpfen ihrer Klasse. (Ausnahme: Überdeckung durch gleichlautende Variablen. Abhilfe: `this`).
- Mit `public` gekennzeichnete Instanzvariablen sind auch ausserhalb sichtbar. Nur in Bibliotheksklassen.

Parameter

- *Parameter* werden in der Parameterliste einer Methodendeklaration deklariert.
- Bei jedem Aufruf ihrer Methode werden sie ins Leben gerufen und mit den konkreten Parametern des Aufrufs initialisiert.
- Ihre Lebensdauer endet mit dem Verlassen der Methode.
- Parameter sind nur sichtbar im Rumpf ihrer Methode. (Ausnahme: Überdeckung durch gleichlautende lokale Variablen. Abhilfe: hier keine)

Lokale Variablen

- *lokale Variablen* werden innerhalb eines Blocks (Methodenrumpf, Block-Statement) deklariert und müssen explizit initialisiert werden.
- Sie werden bei der Abarbeitung ihrer Deklaration ins Leben gerufen; ihre Lebensdauer endet mit dem Verlassen des Blocks in dem sie deklariert wurden.
- Lokale Variablen sind nur in ihrem Block sichtbar. Ausnahme: Überdeckung.

```
int i = 10;
{
  int i = 8;
}
System.out.println(i); // druckt 10
```


Konstruktoren (Wdh.)

Eine Konstruktordefinition hat die Form

NameDerKlasse(typ_1 par_1 , typ_2 par_2 , ... , typ_n par_n) {
    konstruktorRumpf
}

der Konstruktor wird in der Form

$$\texttt{new}\ NameDerKlasse(e_1, \ldots, e_n)$$

aufgerufen.

Das ist ein *Ausdruck* dessen Wert ein frisches Objekt der Klasse *nameDerKlasse* ist.

Konstruktorauswertung

Die Instanzvariablen des neu erzeugten Objekts werden wie folgt initialisiert:

- Zunächst werden sie mit den Defaultwerten (0 für Zahlen, `null` für Objekte) besetzt.
- Dann werden die Ausdrücke $e_1, \ldots, e_n$ ausgewertet und ihre Werte den Parametern par_1, ..., par_n zugewiesen.
  Wie immer müssen die Typen stimmen.
- Schließlich wird der Block *konstruktorRumpf* ausgeführt.
  Typischerweise hat das eine Zuweisung zu den Instanzvariablen in Abhängigkeit von den Parametern zur Folge.

Beispiel

```
Bankkonto(double betrag) {
    kontostand = betrag + 5.0;
}
```

Der Ausdruck `new Bankkonto(`e`)` hat als Wert einen Verweis uaf ein frisches Objekt der Klasse `Bankkonto`, dessen Instanzvariable `kontostand` initialisiert wurde mit dem Wert des Ausdrucks e plus 5.0.

Overloading

Man kann mehrere Konstruktoren in einer Klasse definieren.

Welcher zur Ausführung kommt richtet sich nach Anzahl und Typ der konkreten Parameter beim Aufruf.

Ebenso kann es mehrere Methodendefinitionen mit demselben Methodennamen geben.

Diese Möglichkeit bezeichnet man als Overloading (auch dt. Überladen).

Beispiele

```
public void einzahlen(int betragEuro, int betragCent) {
    kontostand = kontostand + betragEuro + betragCent / 100.0;
}

Bankkonto() {
    kontostand = 5.0;
}
```

Jetzt kann man z.B. schreiben:

```
Bankkonto johannaSpar = new Bankkonto(1000.0);
Bankkonto matthiasGiro = new Bankkonto();

johannaSpar.einzahlen(1000.10);
matthiasGiro.einzahlen(39, 95);
```

Statische Methoden

Man kann in einer Klasse eine Methodendefinition mit dem Qualifikator `static` versehen.

Man hat dann innerhalb der Methode keinen Zugriff auf die Instanzvariablen.

Auf der anderen Seite kann man die Methode dann aufrufen ohne Bezug auf ein bestimmtes Objekt.

Damit man weiß, welche Methode gemeint ist, muss man aber den Klassennamen angeben.

Eine in Klasse *NameDerKlasse* definierte statische Methode *nameDerMethode* ruft man also so auf:

$$NameDerKlasse\texttt{.}nameDerMethode(e_1, \ldots, e_n)$$

Statische Methoden sind vergleichbar den *Funktionen* und *Prozeduren* von Pascal, C, etc.

Man benutzt sie z.B. um mathematische Funktionen zu realisieren.

Beispiel

Man könnte in die Klasse Bankkonto eine statische Methode `markNachEuro` schreiben, die DM-Beträge in € konvertiert:

```
public static double markNachEuro(double markBetrag) {
    return markBetrag / 1.9558; // oder was auch immer
}
```

Man kann dann zum Beispiel schreiben:

```
johannaSpar.einzahlen(Bankkonto.markNachEuro(500.0));
```

Man kann natürlich nicht schreiben `Bankkonto.einzahlen(...)`. Welchen Wert sollte denn dann `kontostand` haben?

`johannaSpar.markNachEuro(...)` ist erlaubt, aber komisch.

“Bibliotheksklassen”

Oft hat man in einer Klasse nur statische Methoden.

So eine Klasse braucht dann keine Instanzvariablen zu haben.

Sie dient einfach der Gruppierung verwandter Operationen.

Zum Beispiel könnten wir eine Klasse `Numeric` schreiben, die numerische Verfahren enthält und insbesondere eine statische Methode für ungefähre Gleichheit.

```
public static approxEqual(double x, double y) {
   final EPSILON = 1E-12;
   return Math.abs(x-y) <= EPSILON
```


Zusammenfassung

- Wdh. Klassendefinition.
- Arten von Variablen, Typ, Lebensdauer, Sichtbarkeit.
- Statische Methoden.