

Objekt

Repräsentation eines Objektes der realen Welt in der Terminologie objektorientierter Programmiersprachen

- besitzen **Attribute** (Eigenschaften), deren Werte i.Allg. veränderlich sind
- reagieren auf an sie gesendete **Botschaften** durch gewisse Aktionen

Klasse

Gesamtheit von Objekten

- mit denselben Attributen,
- die dieselben Botschaften verstehen und auf dieselbe Weise darauf reagieren,
- unterscheiden sich in den Werten ihrer Attribute

Objekte sind *Exemplare (Instanzen)* einer Klasse

Kapselung

Interna von Objekten sind nach außen unsichtbar und können von außen nicht manipuliert werden

Methode

bestimmt das Verhalten eines Objektes auf eine *Botschaft*

Datenelemente (Instanz-Variablen)

Variablen für die Werte der *Attribute* der Objekte

Vererbung

Weitergabe von Merkmalen und Fähigkeiten (Datenelementen und Methoden), wenn neue Klassen aus vorhandenen abgeleitet werden
—> hierarchisches Klassensystem

Polymorphismus

verschiedene Reaktionen von Instanzen verschiedener Unterklassen auf eine gemeinsam verstandene Botschaft
—> Überschreiben von Methoden

Beispiel:

```
void moveTo(int newX, int newY) {  
    x = newX;  
    y = newY;  
}
```

Signatur:

```
void moveTo(int newX, int newY)
```

- Ergebnistyp / Rückgabetyt
- Identifier
- Parameterliste in runden Klammern: kein Semikolon

Rumpf:

- Anweisungsblock in geschweiften Klammern
- enthält die Implementierung der Methode

Ergebnistyp einer Methode

- entweder der *Datentyp* der Variable (oder des Literals), deren (dessen) Wert von der Methode mittels

```
return ⟨Variable⟩;
```

an den Aufrufer zurückgegeben wird

- oder `void` (d.h. es wird kein Ergebnis zurückgegeben).

Parameterliste

Eine durch Kommata getrennte Folge von Variablendefinitionen, die als Liste der *formalen Parameter* bezeichnet wird.

Die formalen Parameter sind Methodenvariablen, die beim Methodenaufruf durch die aktuellen Parameter *initialisiert* werden.

Konstruktor

Methode ohne Ergebnistyp, deren Name mit dem Klassennamen übereinstimmt.

```
Point(int newX, int newY) {  
    // Anweisungen  
}
```

Impliziter Konstruktor

Steht in einer Klasse dann und nur dann zur Verfügung, wenn keine Konstruktoren explizit definiert sind.

Er hat eine leere Parameterliste .

```
Point() {  
    // Anweisungen  
}
```

Standardinitialisierung von Datenelementen

Alle Datenelemente, die nicht durch die Parameter des Konstruktors initialisiert werden, erhalten standardmäßige Initialwerte wie folgt:

byte, short, int	0
long	0L
float	0.0f
double	0.0d
char	'\u0000'
boolean	false
Verweistypen	null

Methodenvariablen hingegen müssen immer per Anweisung initialisiert werden!

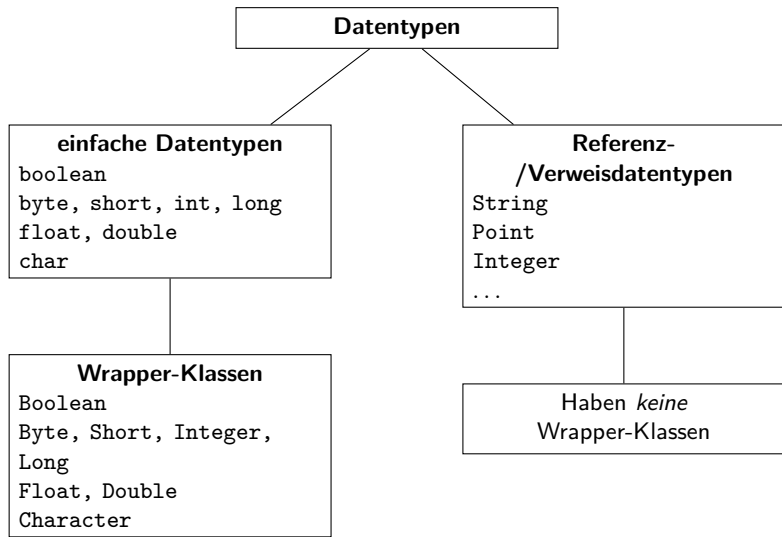
Überladen von Konstruktoren und Methoden

Vereinbarung mehrerer Konstruktoren/Methoden mit demselben Methodennamen, aber verschiedenen Parameterlisten

```
void moveTo(int newX, int newY)
```

```
void moveTo(int newX)
```

- In jeder Klasse darf es keine zwei Konstruktoren/Methoden mit identischem Bezeichner *und* identischer Parameterliste geben.
- Beim Methodenaufruf wird jene Methode angesprochen, deren Liste formaler Parameter zu der Liste der aktuellen Parameter im Methodenaufruf passt.
- Mit einer anderen Parameterliste darf auch ein anderer Ergebnistyp vereinbart werden.



Datentypen (2)

Einfache Datentypen

```
int num1, num2;  
boolean b1, b2;
```

```
num1 = -12;  
num2 = 4;  
b1 = true;  
b2 = false;
```

num1	<input type="text" value="-12"/>
num2	<input type="text" value="4"/>
b1	<input type="text" value="true"/>

Referenz-/Verweisdatentypen

Variablendefinition

```
Point p1, p2;  
String str1, str2;
```

Variableninitialisierung

```
p1 = null;  
p2 = new Point();  
str1 = new String("Hallo");  
str2 = "Hallo?";
```

im Hauptspeicher

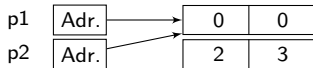
p1	<input type="text" value="null"/>	x	y
p2	<input type="text" value="Adr."/>	→	<input type="text" value="0"/> <input type="text" value="0"/>
str1	<input type="text" value="Adr."/>	→	<input hallo\""="" type="text" value="\"/>

Datentypen (3)

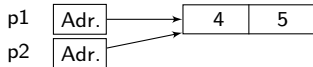
```
p1 = new Point();  
p2.moveTo(2,3);
```



```
p2 = p1;
```



```
p1.moveTo(4,5);
```



```
num2 = num1;
```



```
num1 = 99;
```



Schlüsselwörter zur Zugriffsmodifikation

- Datenelemente/Methoden/Konstruktoren mit dem Modifier

<code>public</code>	sind für alle Klassen sichtbar;
<code>private</code>	sind nur für die Klasse sichtbar, in der sie vereinbart sind;
<i>ohne Modifier</i>	sind für Klassen aus demselben Paket (Verzeichnis) sichtbar

- Der Modifier ist das *erste* Schlüsselwort in der Definition/Signatur.
- Klassen dürfen auch `public` sein. (`public class ...`)
- Diese Schlüsselwörter dürfen in der Definition von Methodenvariablen **nicht** auftreten!

Eine Java-Applikation ist eine Klasse, in der die Methode `main` definiert ist:

```
public static void main(String[] args) {  
    ...  
}
```

Es können weitere Klassen benutzt werden.

- genau eine `main`-Methode pro Applikationsklasse
- Hier startet das Programm.

- Unterscheidung von Groß- und Kleinschreibung
- Bezeichner müssen mit einem Buchstaben, `_` oder `$` beginnen.
- sprechende Bezeichner (Ausnahme: Schleifenzähler u.ä.)

Grundsätze für Bezeichner

einfache Datentypen	vollständig klein	<code>int</code>
Referenzdatentypen/ Klassen	große Anfangsbuchstaben jedes Teilwortes	<code>String</code> , <code>LayoutManager</code>
Variablen und Methoden	große Anfangsbuchstaben außer beim ersten Teilwort	<code>name</code> <code>readLine()</code>
Konstanten	vollständig groß Unterstriche	<code>PI</code> <code>CIRCLE_RADIUS</code>