Grundelemente objektorientierter Sprachen (1)

- **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

Grundelemente objektorientierter Sprachen (2)

- **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

  $\longrightarrow$ hierarchisches Klassensystem
- **Polymorphismus**
  verschiedene Reaktionen von Instanzen verschiedener Unterklassen auf eine gemeinsam verstandene Botschaft

  $\longrightarrow$ Überschreiben von Methoden

Methoden

Beispiel:

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

Signatur:

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

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

Rumpf:

- Anweisungsblock in geschweiften Klammern
- enthält die Implementierung der Methode

Ergebnistypen/Parameter von Methoden

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

    `return` ⟨*Variable*⟩`;`

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

- Die Parameterliste ist (syntaktisch) eine durch Komma 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.

Konstruktoren

- Konstruktoren sind Methoden ohne Ergebnistyp, deren Name mit dem Klassennamen übereinstimmt.

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

- Der implizite 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

einfache Datentypen

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

Wrapper-Klassen:

```
Boolean
Byte, Short, Integer, Long
Float, Double
Character
```

Referenz-/Verweisdatentypen

```
String
Point
Integer
...
```

haben KEINE Wrapper-Klassen

einfache Datentypen

Referenz-/Verweisdatentypen

Variablendefinition

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

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

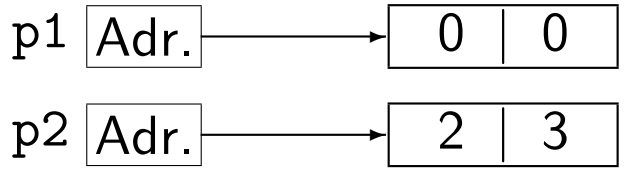
Variableninitialisierung

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

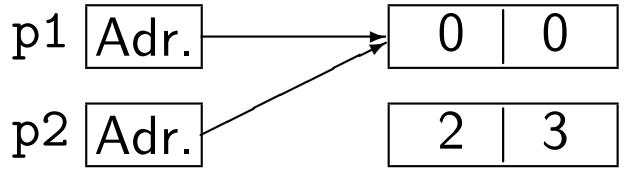
```
p1 = null;
p2 = new Point();
str1 = new String("Hallo!");
str2 = "Hallo?";
```

im Hauptspeicher:

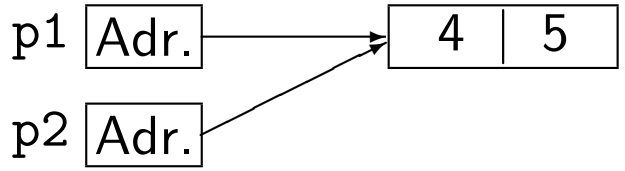
```
p1 = new Point();
p2.moveTo(2,3);
```



```
p2 = p1;
```



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



```
num2 = num1;
```

```
num1 = 99;
```

Schlüsselwörter zur Zugriffsmodifikation

- Datenelemente/Methoden/Konstruktoren mit dem Modifier

`public`	sind für alle Klassen sichtbar;
`private`	sind nur für die Klasse sichtbar, in der sie vereinbart sind;
ohne Modifier	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!

Klassen, Applikationen und die `main`-Methode

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.

Bezeichner und Namenskonventionen

- 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	`int`
– Referenzdatentypen/ Klassen	große Anfangsbuchstaben jedes Teilwortes	`String`, `LayoutManager`
– Variablen und Methoden	große Anfangsbuchstaben außer beim ersten Teilwort	`str` `readLine`
– Konstanten	vollständig groß	`PI`