Kapitel 6: Objektorientierung

- Imperative Programmierung
  - Beschreibung von Abläufen (Algorithmen).
  - Fehlt noch: Beschreibung von Datenstrukturen (Bsp. Array).
  - Daten- und Ablaufbeschreibungen meist getrennt.

- Objektorientierte Programmierung
  - Enge Zusammenführung von Daten- und Ablaufbeschreibungen.
  - Objekte der realen Welt werden durch Objektklassen beschrieben.
  - Die Objekte sind selbstständige Struktur- und Funktionseinheiten.
  - Im Programmablauf interagieren die Objekte miteinander.

Klassen und Objekte

- Klassen
  - Klassen sind die grundlegenden Einheiten der Programmierung in Java.
  - Ein Java-Programm besteht aus einer Menge von Klassen.
  - Klassen beschreiben die Struktur und das Verhalten von Objekten.
  - Eine Klasse ist eine *statische* Objektbeschreibung (= Programmtext).

- Objekte
  - Objekte werden *dynamisch* im Programmablauf erzeugt und über Referenzen angesprochen und weitergegeben.
  - Jedes Objekt ist eine Ausprägung (Exemplar, Instanz) einer bestimmten Klasse, die Klasse ist der Typ des Objekts.
  - Mit dem booleschen Operator *instanceof* lässt sich die Zugehörigkeit eines Objekts *p* zu einer Klasse *P* prüfen:

    if (*p* instanceof *P*) …

Zustand und Verhalten von Objekten

- Ablauf objektorientierter Programme
  - Erzeugung von Objekten durch Instantiierung von Klassen.
  - Objekte interagieren durch Mitteilungen in Form gegenseitiger Aufrufe.
- Verhalten von Objekten
  - Das Verhalten eines Objekts wird durch die Methoden seiner Klasse beschrieben.
  - Das Verhalten eines Objekts hängt von seinem aktuellen Zustand sowie von den aktuellen Parametern des Aufrufs ab.
- Zustand von Objekten
  - Objekte haben einen Zustand, der in den Attributen gespeichert wird.
  - Der Zustand ergibt sich aus der Historie eines Objekts, d.h. welche Methoden mit welchen Parametern für das Objekt aufgerufen wurden (abgesehen von direkten Zugriffen auf die Attribute).

Attribute

- Attribute (*fields*)
  - speichern *Zustand* eines Objekts.
  - stellen die Datenstruktur von Objekten einer Klasse dar.
  - Jedes Attribut hat einen Typ.
- Beispiel

```
class MyClass {
    float scale;
    boolean valid;
    String description;
}
```

- Zugriff über Punktnotation

  allgemein: Objektvariable "." Attribut

  Bsp. (für MyClass mc): mc.valid

- Attributtypen
  - *einfache Datentypen* (boolean, char, int, double, etc.): Attribute enthalten *Werte*.
  - *Objekttypen* (auch Arrays): Attribute enthalten *Referenzen* auf bezogene Objekte.
  - die prinzipiell möglichen Zustände eines Objekts ergeben sich aus dem kartesischen Produkt der Wertebereiche der Attribute.

    Bsp. MyClass:

    float $\times$ boolean $\times$ String

Methoden

- Methoden (*methods*)
  - legen das *Verhalten* von Objekten einer Klasse fest.
  - stellen die funktionalen bzw. prozeduralen Komponenten von Objekten dar.
  - bestehen aus Anweisungen, die beim Aufruf ausgeführt werden.
  - Methodenaufrufe stellen Mitteilungen an (andere) Objekte dar.

- Deklaration
  - Methoden für Objekte werden ohne **static** deklariert.

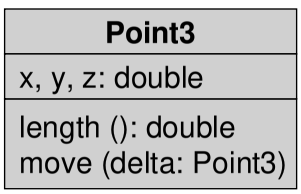
```
class MyClass {
  …
  void invalidate () {valid = false;}
}
```

- Aufruf von Methoden
  - Punktnotation (wie bei Attributen): mc.invalidate();

Beispiel: Punkte (Vektoren) im 3D

Klassendiagramm in UML
(*Unified Modeling Language*)

oder



Name
Attribute
Methoden

Notation in Java

```
public class Point3 {
    /** Cartesian Coordinates */
    public double x, y, z;

    /** distance to origin */
    public double length() {
      return Math.sqrt (x*x + y*y + z*z);
    }

    /** translation by vector p */
    public void move (Point3 p) {
      x += p.x;  y += p.y;  z += p.z;
    }
}
```

Erzeugung von Objekten

- Erzeugung eines Objektes durch **new**
  - Reservierung von Speicherplatz auf dem Heap.
  - Rückgabe der Adresse als Objektreferenz.

```
Point3 p = new Point3();
```

- Phasen der Objekterzeugung
  - *Deklaration* Variable oder Attribut für Objektreferenz vereinbaren.
  - *Instantiierung* neues Objekt durch **new** erzeugen.
  - *Initialisierung* Objekt in definierten Ausgangszustand bringen.

Initialisierung von Objekten

- Implizite Initialbelegung von Attributen

Typ	**Anfangswert**
Wahrheitswert (*boolean*)	**false**
Zeichen (*char*)	'\u0000'
ganze Zahl (*byte, short, int, long*)	0
Gleitpunktzahl (*float, double*)	0.0f bzw. 0.0d
Objektreferenz (auch *String, Array*)	**null**

- Explizite Initialbelegung von Attributen

```
class Sphere {
    float radius = 1.0f;
    …
}
```

Konstruktoren

- Begriff
  - Methoden zur Initialisierung
  - heißen genauso wie die Klasse.
  - haben keinen formalen Ergebnistyp.
  - werden implizit bei **new** aufgerufen, expliziter Aufruf ist nicht möglich.
- Erlauben komplexe Initialisierungen
  - falls einfache Initialbelegung nicht ausreicht, oder
  - falls Parameter benötigt werden.
- Regeln
  - Eine Klasse kann mehrere Konstruktoren mit verschiedenen Parametern haben.
  - Impliziter Konstruktor ohne Parameter existiert nur, falls sonst kein Konstruktor definiert ist.

- Beispiel

```
public class Point3 {
  …
  public Point3 (
    double xx, double yy, double zz) {
    x = xx; y = yy; z = yy;
  }
  public Point3 () {
    x = 0.0; y = 0.0; z = 0.0;
  }
}
```

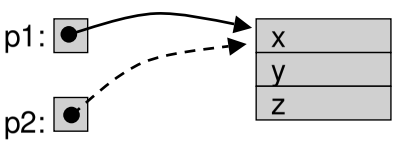
Aufruf mit aktuellen Parametern:

```
Point3 p1 = new Point3();   // is (0,0,0)
Point3 p2 = new Point3(1.3, 7.0, 2.6);
```


Zuweisung und Übergabe von Objekten

- Zuweisungen von Objekten
  - Nur erlaubt, wenn Typen gleich (Strukturäquivalenz reicht nicht, Namensäquivalenz nötig).
  - Leere Referenz **null** ist mit allen Typen kompatibel.
  - Es wird nur die Referenz, nicht jedoch der Inhalt kopiert.

```
Point3 p2 = p1;
```



- Objekte als Parameter
  - Als aktuelle Parameter werden nur Referenzen auf Objekte kopiert, nicht die Objekte selbst (Übergabe per Referenz, *call by reference*).
  - Folge: Änderungen von Objekten werden auf Originalen wirksam!

- Duplizieren von Objekten

  Falls nötig, können mit der Methode *clone*() Duplikate von Objekten erzeugt werden.

```
Point3 p2 = p1.clone();
```

Vergleiche von Objekten

- Vergleich von Objekten
  - Vergleichsoperatoren == und != vergleichen nur Objektreferenzen.
  - Operatoren <, <=, >, >= sind für Objekte nicht definiert.
  - Wertevergleich durch implizit vordefinierte Methode *equals*.
  - Semantik von *equals*: attributweiser Wertevergleich.
  - Explizite Redefinition von *equals*, falls komplexere Semantik nötig.

- Beispiel

```
public class Point3 {
    …
    public boolean equals (Point3 p) {
      return (x == p.x) && (y == p.y) && (z == p.z);
    }
}
```


Lebensdauer von Objekten

- Lebensdauer von Objekten
  - Objekte entstehen durch Erzeugung mit **new** (*Instantiierung*).
  - Lebensdauer erlischt, falls Objekt nicht mehr erreichbar ist.

- Speicherbereinigung
  - Speicherplatz von erloschenen Objekten wird bei automatischer Speicherbereinigung (*garbage collection*) freigegeben.
  - Explizite Freigabe von Objekten ist nicht möglich.

- Bewertung
  - \+ lebende Objekte können nicht aus Versehen gelöscht werden.
  - \+ Programmierer müssen sich nicht um Speicherverwaltung kümmern.
  - – automatische Speicherbereinigung kann zu ungünstiger Zeit starten.

Spezielle Objektreferenzen

- Leere Referenz: **null**
  - Konstante **null** bezeichnet leere Referenz, kein Zugriff möglich.
- Empfängerobjekt: **this**
  - **this** ist Referenz auf aktuelles Objekt einer Methode (Empfänger der Mitteilung).
  - In Methoden der eigenen Klasse ist Zugriff über **this** möglich, aber meist unnötig:

    valid statt **this**.valid
    length() statt **this**.length()

  - **this** ist nötig, falls aktuelles Objekt als Parameter übergeben wird:

    p.move (this);

  - **this** ist nötig, falls Attribute des aktuellen Objekts verschattet sind:

```
public Point3 (double x, double y, double z) {
  this.x = x;  this.y = y;  this.z = z;
}
```


Klassenattribute und Klassenmethoden

- Klassenattribute
  - werden mit **static** deklariert.
  - existieren nur einmal, unabhängig von Objekten („globale Variable“).
  - liegen im Namensraum der Klasse, Zugriff ggf. über Punktnotation.
- Klassenmethoden
  - werden mit **static** deklariert.
  - können nicht auf **this** zugreifen, da sie unabhängig von Objekten sind.
  - liegen im Namensraum der Klasse, Aufruf ggf. über Punktnotation.
- Klasseninitialisierer
  - unbenannte **static**-Methode.
  - Initialisierung von Klassenattributen.
- Beispiel

Deklarationen

```
class Point3 {
    static float gravity = 9.81f;
    static int count;
    static { count = 0; }
    …
}
```

Zugriff / Aufruf

```
float g = x / y * Point3.gravity;
System.out.println (Point3.count);
```