- Objektorientiertes Programmieren mit Java -

Überblick

- Einführung in das Programmieren (S. 1-140, S. 181-226, = 185 Seiten)
  - Grundlagen (Kapitel 1, im Schnelldurchlauf!)
  - Datentypen und Operatoren (Kapitel 2)
  - Kontrollanweisungen (Kapitel 3)
  - Arrays (Kapitel 5)
- Objektorientierte Programmierung (S. 141-180, S. 227-388, = 200 Seiten)
  - Klassen, Objekte Methoden (Kapitel 4)
  - Genauere Betrachtung der Methoden und Klassen (Kapitel 6)
  - Vererbung und Polymorphie (Kapitel 7)
  - Pakete und Schnittstellen (Kapitel 8)
  - Ausnahmebehandlung (Kapitel 9)

Ziele

- Wiederholung: Klasse vs. Objekt
- Bestandteile und Aufbau einer Klasse
- Erzeugen einer Instanz einer Klasse
- Objektverweise
- Methoden
  - Allgemeines zur Methode
  - Rückgabe eines Wertes
  - Verwenden von Parametern
- Konstruktoren
- Objektverweis "this"
- Garbage Collection

Wiederholung

- Klasse als
  - Grundlage der OOP
  - Bauplan für Objekte
  - logische Abstraktion

- Objekt als
  - Instanz einer Klasse
  - "physisches" Ding

- Bestandteile einer Klasse
  - Attribute und Methoden
  - nur A oder nur M möglich, meistens aber beides

Definition einer Klasse (1)

```
class Klassenname {

    // Deklaration der Attribute
    Datentyp Variablenname;
    Datentyp Variablenname;
    ...

    // Deklaration der Methoden
    Datentyp Methodenname(Parameterliste) {
        ...
    }

    Datentyp Methodenname(Parameterliste) {
        ...
    }
    ...
}
```

Definition einer Klasse (2)

- Fragestellungen
  - welche Attribute und Methoden sind von Interesse?
  - was kommt in die Klasse rein, was kommt woanders hin?
  - was wird im Programm überhaupt nicht abgebildet?
  - welche Datentypen sind sinnvoll?

```
class Kfz {

    // Deklaration der Attribute
    int sitzplaetze;
    double tankinhalt;
    double verbrauch;

    // Deklaration der Methoden
    ...
}
```

Die Definition einer Klasse erzeugt einen gleichnamigen Datentyp.

2. Axiom von Grabo

Klasse -> Datentyp

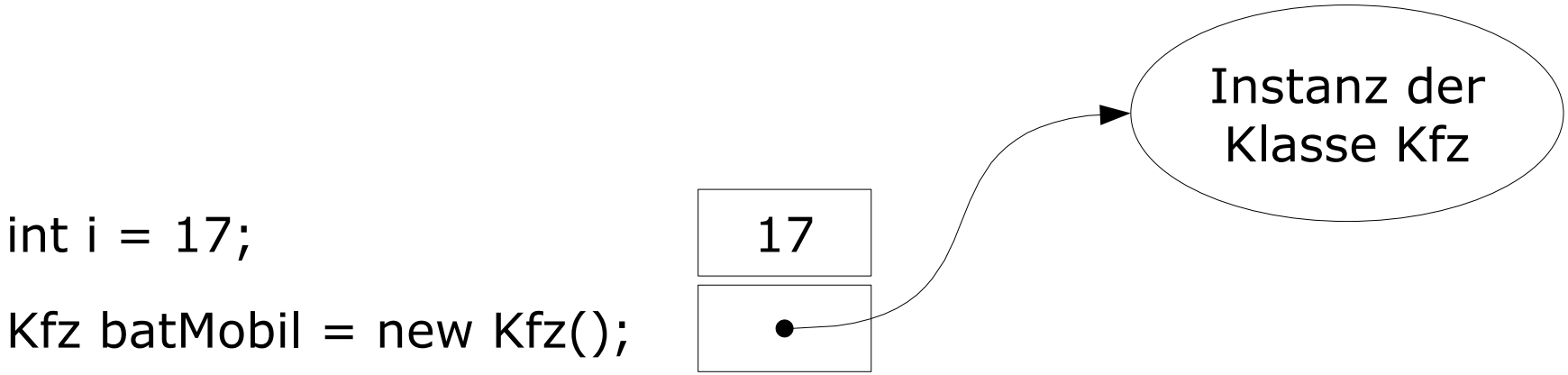
- Definition einer Klasse erzeugt gleichnamigen Datentyp
- so können Variablen, die Objekte von diesem Datentyp aufnehmen, deklariert werden

```
class Kfz {
    int sitzplaetze;
    double tankinhalt;
    double verbrauch;
}

class KfzDemo {
    public static void main(String[] args) {
        // Variable vom Datentyp(!) Kfz
        Kfz batMobil;
        ...
    }
}
```

Erzeugen einer Instanz einer Klasse

- Erzeugen einer Instanz mit *new*
- Variable speichert den *Verweis*!
- ... und nicht das Objekt

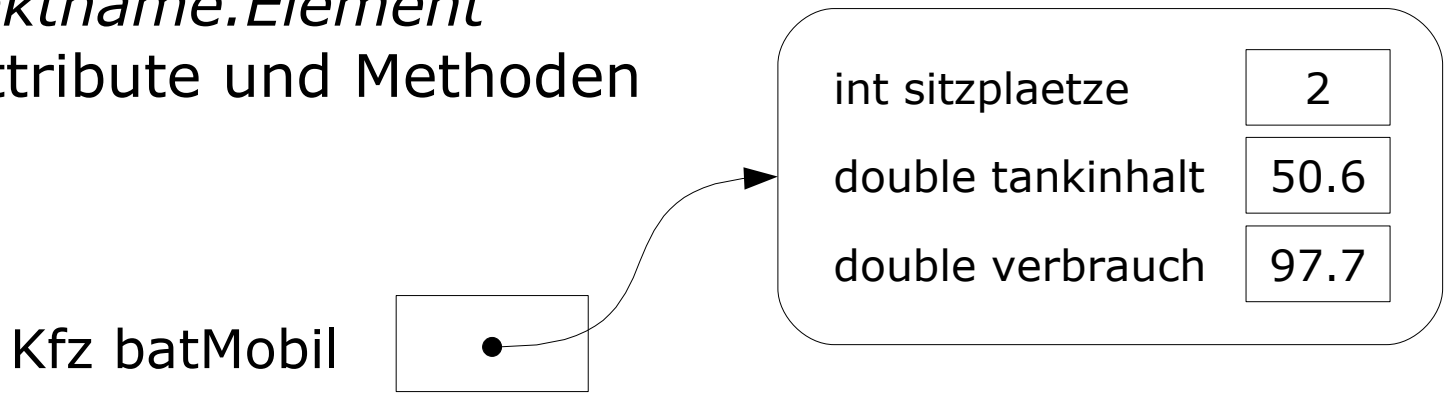



```
int i; // primitiver Datentyp
Kfz batMobil; // Objektdatentyp

i = 17; // Wert wird hier direkt gespeichert
batMobil = new Kfz(); // Verweis auf neu erzeugtes Objekt
```

Zugriff auf das erzeugte Objekt

- Dereferenzierungsoperator . (Punkt)
- Syntax: *Objektname.Element*
- Zugriff auf Attribute und Methoden




```
Kfz batMobil = new Kfz();

// Zugriff auf Attribute des Objekts batMobil
batMobil.sitzplaetze = 2;
batMobil.tankinhalt = 50.6;
batMobil.verbrauch = 97.7;

double reichweite = (batMobil.tankinhalt / batMobil.verbrauch) * 100;
```

Methoden: Allgemeines

- Aufgaben von Methoden hauptsächlich
  - Manipulieren von Daten eines Objekts
  - Zugriff auf Daten/Information eines Objekts

- Interaktion von Objekten durch Methodenaufrufe
  - Aktionen werden angestossen
  - Ergebnisse von Berechnungen werden weiterverwendet

- Methode enthält Anweisungen
  - ... die beim Aufruf der Methode ausgeführt werden

```
// Allgemeine Form
Datentyp Methodenname(Parameterliste) {
    Anweisung;
    ...
}
```

Methoden: Aufruf

- Beispiel für Methode und deren Aufruf
- Hinweis: innerhalb der Klasse keine Deref. mehr notwendig

```
class Kfz {
    int sitzplaetze;
    double tankinhalt;
    double verbrauch;

    // Ausgabe der Reichweite
    void gibReichweiteAus() {
        double r = (tankinhalt / verbrauch) * 100;
        System.out.println("Reichweite: " + r + " km");
    }
}

// Aufruf der Methode,
// z.B. aus der main() Methode in KfzDemo
batMobil.gibReichweiteAus();
```

Methoden: Werte zurückliefern

- Methode ohne Rückgabewert
  - Methode endet entweder nach der letzten Anweisung,
  - oder schon vorher mittels return – Anweisung

- Methode mit Rückgabewert
  - per return wird ein Wert zurückgegeben

```
class Kfz {
    double berechneReichweite() {
        double ergebnis = (tankinhalt / verbrauch) * 100;
        return ergebnis;
    } //...
}

// Aufruf zum Beispiel:
double reichweite = batMobil.berechneReichweite();
System.out.println("Reichweite: " + reichweite + " km");
```

Methoden: Parameter verwenden (1)

- Argument
  - eine Methode kann beim Aufruf einen oder mehrere Werte verlangen. Diese Werte nennt man Argumente.

- Parameter
  - die Variablen, die innerhalb der Methode diese Werte aufnehmen, nennt man Parameter.

```
int berechneSumme(int a, int b) {
    int summe = a + b;
    return summe;
}

int x = berechneSumme(4, 3);
int y = berechneSumme(2, x);
int z = berechneSumme(y, berechneSumme(x, 5));
```

Methoden: Parameter verwenden (2)

```
class Kfz {
    double berechneVerbrauch(double distanz) {
        double v = (distanz / 100) * verbrauch;
        return v;
    }
    // Rest der Klasse...
}

class KfzDemo {
    public static void main(String[] args) {
        Kfz batMobil = new Kfz();
        // Attributwerte setzen...

        double d = 120.7;
        double b = batMobil.berechneVerbrauch(d);
        System.out.println("Auf " + d + " km verbraucht das " +
            " Batmobil " + b + " Liter Benzin.");
    }
}
```

Kunstruktoren (1)

- Möglichkeit, Objekt bei Erzeugung zu initialisieren
- syntaktisch ähnlich einer Methode, allerdings:
  - Konstruktor heisst immer wie die Klasse
  - kein expliziter Rückgabetyp
- wird bei Erzeugung einer Instanz (zB "new Kfz()") ausgeführt

```
class Kfz {
    // ...
    Kfz() {
        sitzplaetze = 4;
        tankinhalt = 0.0;
        verbrauch = 10.0;
    }
}
```

Kunstruktoren (2)

- It's a Good Thing™
  - neue Instanz kann auf Verwendung vorbereitet werden
  - sinnvolle Initialwerte für Instanzvariablen

```
class KfzDemo {
    public static void main(String[] args) {

        // neue Instanz der Klasse Kfz
        // Konstruktor Kfz() wird ausgeführt
        Kfz k = new Kfz();

        // Ausgabe: 4
        System.out.println(k.sitzplaetze);
    }
}
```

Kunstruktoren (3)

- falls kein Konstruktor vorhanden: Default-Konstruktor
  - Instanzvariablen werden auf 0 bzw. *null* gesetzt

- parametrisierte Konstruktoren möglich,
- mehrere Konstruktoren möglich

```
class Kfz {
    // ...
    Kfz(int s, double t, double v) {
        sitzplaetze = s;
        tankinhalt = t;
        verbrauch = v;
    }
}

Kfz k = new Kfz(4, 40.0, 10.0);
```

Angabe zur Übung (1)

- das folgende Programm...

```
class KfzDemo {
    public static void main(String[] args) {
        // Sitzplätze, Tankinhalt, Verbrauch
        Kfz k = new Kfz(3, 10.0, 12.4);

        k.anlassen();
        k.fahre(8.42);
        k.einsteigen("Frida");
        k.einsteigen("Hubert");
        k.einsteigen("Kurt");
        k.einsteigen("Lanzelott");
        k.fahre(10.12);
        k.anlassen();
        k.fahre(12.14);
        k.fahre(8.34);
        k.fahre(228.34);
        k.abstellen();
    }
}
```

Angabe zur Übung (2)

- ... soll die folgende Ausgabe erzeugen.
- Implementiere die Klasse Kfz!

```
Kein/e Fahrer/in!
Kein/e Fahrer/in!
Frida steigt ein.
Frida ist Fahrer/in.
Hubert steigt ein.
Kurt steigt ein.
Kein Platz im Auto für Lanzelott.
Motor läuft nicht!
Motor wird angelassen.
Das KFZ soll 12.14 km fahren.
Dafür braucht es 1.50536 Liter.
Frida fährt 12.14 km.
Im Tank sind noch 8.49464 Liter.
Das KFZ soll 8.34 km fahren.
Dafür braucht es 1.03416 Liter.
Frida fährt 8.34 km.
Im Tank sind noch 7.4604800000000004 Liter.
Das KFZ soll 228.34 km fahren.
Dafür braucht es 28.314159999999998 Liter.
Zu wenig Sprit! Im Tank sind nur 7.4604800000000004 Liter.
Motor wird abgestellt.
```

Java Code Conventions beachten!

- "*CamelNotation*": `DatabaseConnection`, `ArrayList`, ...
- Klassen und Konstruktoren beginnen mit Grossbuchstaben
- Variablen und Methoden beginnen mit Kleinbuchstaben

- Sprechende Namen verwenden

- Nomen für Klassen und Instanzvariablen: `Kfz.verbrauch`
- Verben für Methoden: `berechneVerbrauch(double distanz)`

- Einrückung beachten

- siehe http://java.sun.com/docs/codeconv