



# Programmierungsmethodik 1

# Programmiertechnik

## Klassen

- while-Schleifen
- do-while-Schleifen
- break & continue
- for-Schleifen
- Sichtbarkeitsbereiche

Ausblick für heute

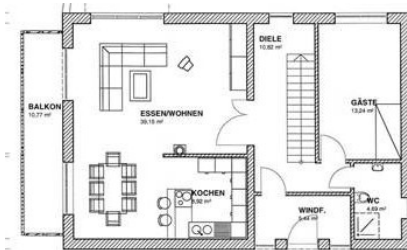
- Ich möchte ein "Ding" aus der realen Welt, das mehrere verschiedenen Eigenschaften hat, in einem Programm repräsentieren.

- Klassen und Objekte
- Referenztypen
- Objektvariablen
- Vergleich und Lebensdauer
- UML

Klassen und Objekte

## Klasse

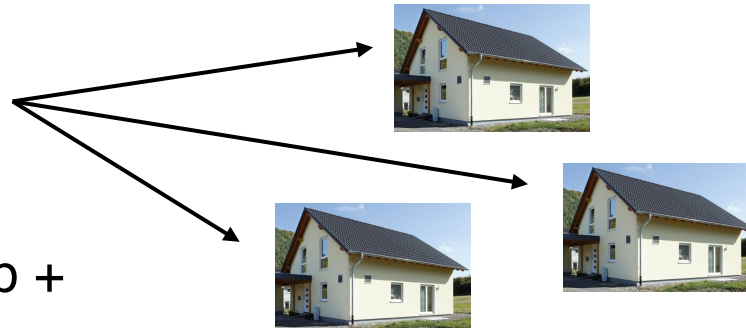
- Bauplan



- hat Eigenschaften aus Typ + Name (Objektvariablen)
- kann Nachrichten empfangen (Methoden)

## Objekt/Instanz

- entsprechend Bauplan gebaut
- zur Laufzeit des Programm



- Eigenschaften haben konkrete Werte

- bisher: einfache (primitive) Datentypen:
  - `byte`, `short`, `int`, `long`, `float`, `double`, `char`, `boolean`
  - → in Java fest vordefiniert!
- jetzt: durch eigene Klassen werden neue Datentypen definiert
  - Definition einer Klasse = Definition eines neuen Typs!



- Datentyp zur Darstellung von Brüchen

```
/**
 * Ein Bruch besteht aus einem Zähler und einem Nenner.
 */
class Bruch {

    /**
     * Zähler.
     */
    int zaehler;

    /**
     * Nenner.
     */
    int nenner;
}
```

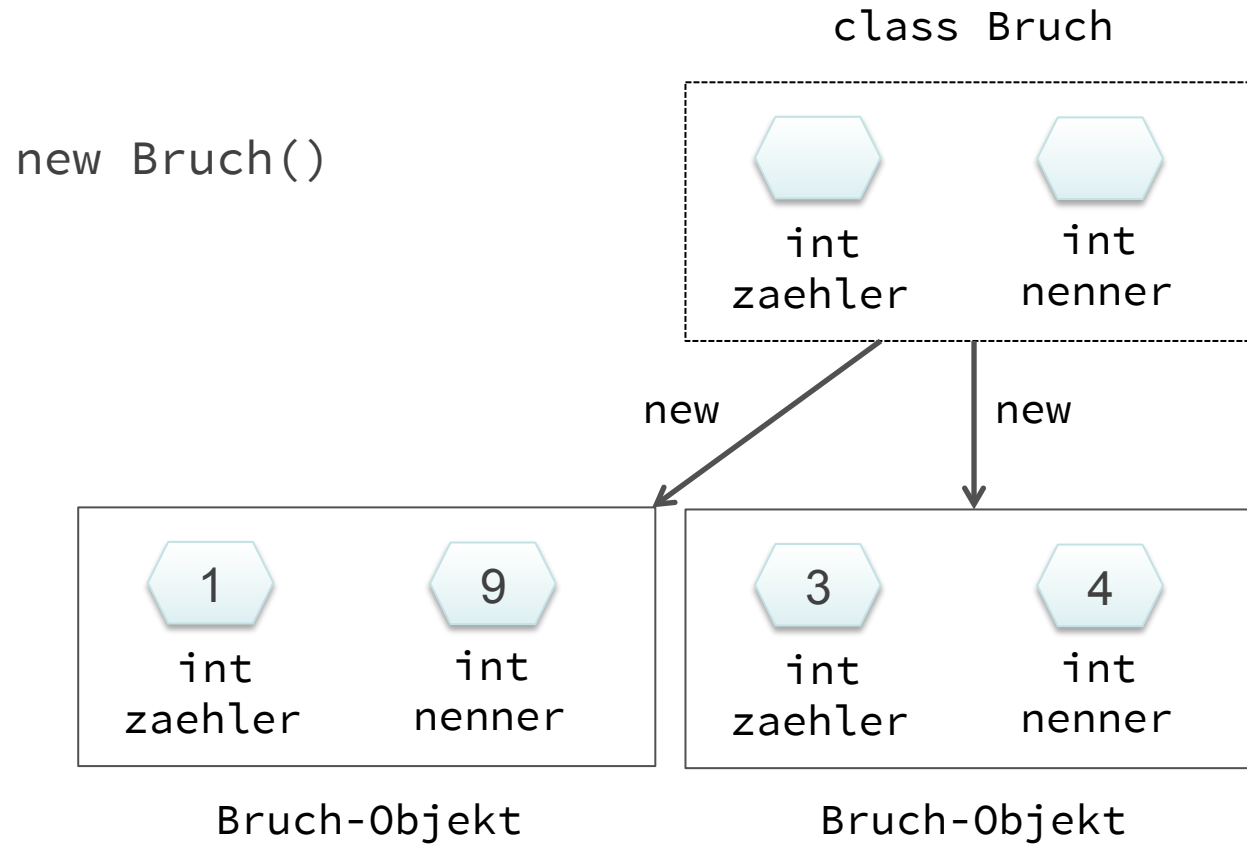
- Klassen sind mit eindeutigen Bezeichnern benannt
- In der Regel Substantive, erster Buchstabe groß
- Syntax:

```
class <Klassenname> {  
    ...  
}
```
- Beispiel: Klasse `Bruch` zur Darstellung von Brüchen

```
class Bruch {  
    ...  
}
```
- jede Klassendefinition sollte in einer eigenen Quelltextdatei stehen
- *Dateiname*  $\Rightarrow$  *<Klassenname>.java*
  - Klasse: `Bruch`
  - Dateiname: *Bruch.java*

- Erzeugen eines neuen Objektes: Operator `new`
  - auch instanziiieren, konstruieren, allokkieren
- Syntax:  
`new <Klassenname>()`
- Beispiel:  
`new Bruch()`
- `new` produziert aus einer Klassendefinition ein einzelnes, neues Objekt dieser Klasse
- mehrere Objekte  $\Rightarrow$  mehrere Aufrufe von `new` nötig

# Erzeugen von Objekten



- new ist unärer Operator
  - Operator: new
- Priorität sehr hoch
  - wie bei anderen unären Operatoren
- Operand von new: Klassenname + leere, runde Klammern
  - Operand: Bruch()
- Wert (Ergebnis) eines new-Ausdrucks:
  - ein neu erzeugtes Objekt vom Typ `<Klassenname>`

- Klassendefinition ist
  - Bauplan
  - Konstruktionsvorschrift
  - Schema
- Objekte der Klasse müssen explizit geschaffen werden, entstehen nicht von alleine
- Objekt = Exemplar, Instanz (mit eigenen Objektvariablen)
- eine Klassendefinition erlaubt beliebig viele Objekte

- Beispiel: Erzeugen eines Bruch-Objektes in einer Anwendungsklasse

```
/**
 * Anwendungsklasse für Brüche.
 *
 */
public class BruchAnwendung {

    /**
     * Programmeinstiegs-Methode.
     */
    public static void main(String[] args) {
        new Bruch();

        ...
    }
}
```

- generell: ausführbare Klassen (Programme) benötigen folgende Methode

```
public static void main(String[] args)
```

- Erstellen Sie eine Klasse `Tier`.
- Schreiben Sie eine `main()`-Methode, in der Sie zwei Instanzen der Klasse `Tier` erzeugen.

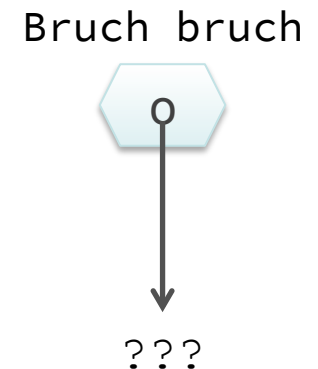
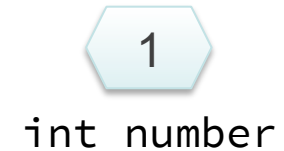


Referenztypen

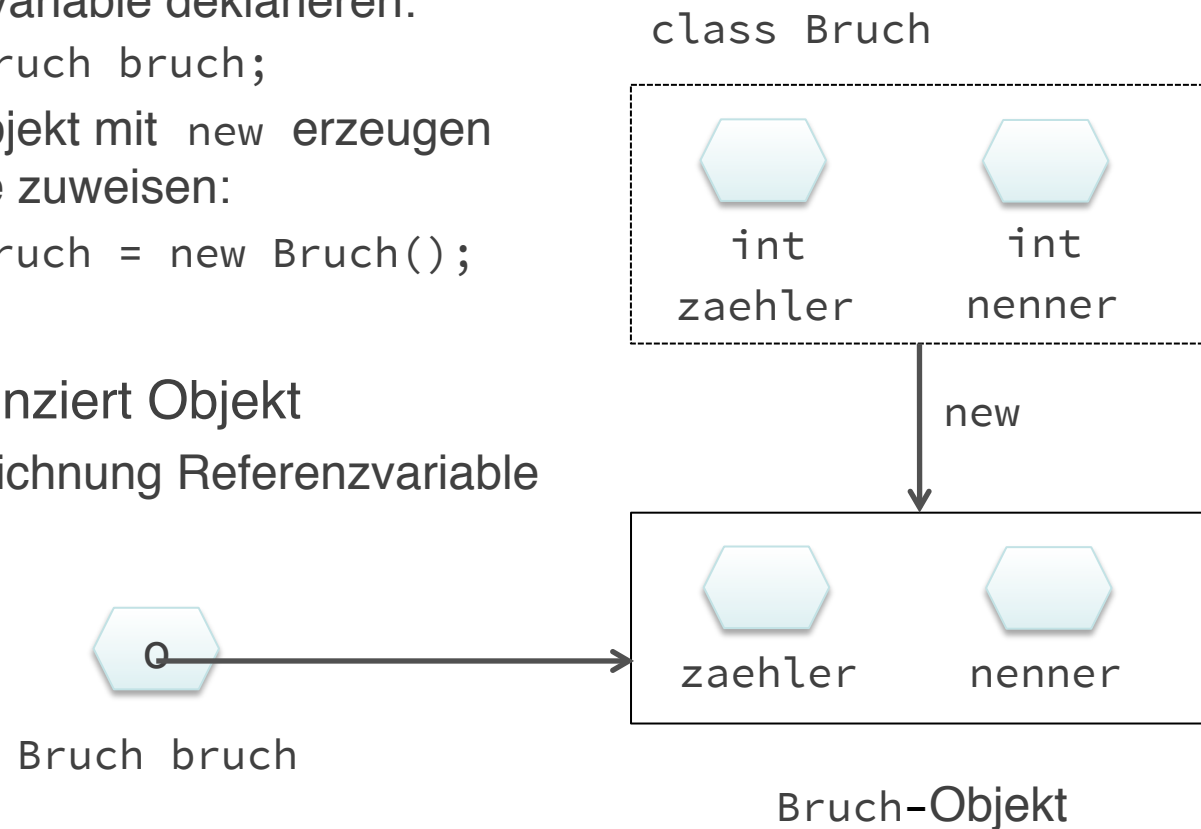
- Bruch = neuer Typ
  - gleichberechtigt neben `int`, `double`, `boolean` etc.
- `int`, `double`, `boolean` etc. sind einfache (primitive) Typen:
  - atomar (nicht unterteilbar)
- Gegensatz: `Bruch` ist ein Referenztyp
  - enthält separate Bestandteile, diese können einzeln angesprochen und verarbeitet werden
- jede Klasse definiert einen eigenen Referenztyp
- Auswahl primitiver Typen liegt fest
  - können nicht neu definiert werden
- erster Nutzen von Klassen: Bündeln der Bestandteile
  - im Beispiel `Bruch`: Zähler und Nenner eines Bruchs bleiben immer zusammen

- Klassen definieren Referenztypen
- alle Typen sind für Variablendeklarationen erlaubt
- Referenzvariable = Variable für einen Referenztyp
  - Zeiger auf Objekt
- Wert einer Referenzvariablen ist ein Objekt
- Gegensatz: „primitive Variable“ = Variable für einfachen (primitiven) Typ

- primitive Variable:
  - Variable und Speicherplatz für Wert untrennbar gekoppelt
  - einfacher Datentyp
  - Beispiel: `int n = 23;`
- Referenzvariable:
  - Variable (Zeiger) und Wert (Objekt) existieren unabhängig und getrennt
  - Deklaration einer Referenzvariablen erzeugt nur die Variable, kein Objekt
  - Beispiel: `Bruch bruch;`
    - Wert ist unbekannt, da nicht initialisiert



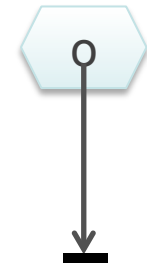
- Deklaration und Initialisierung
  - 1. Referenzvariable deklarieren:  
`Bruch bruch;`
  - 2. Neues Objekt mit `new` erzeugen und Variable zuweisen:  
`bruch = new Bruch();`
- Variable referenziert Objekt
  - daher: Bezeichnung Referenzvariable



- `null` steht für "kein Objekt"
- `null` kann an jede Referenzvariable zugewiesen werden:  
`Bruch bruch = null;`
- `null` = wohldefinierter Wert, kann verglichen werden Beispiel:

```
if ( bruch == null ){
    System.out.println("no object");
}
```
- neu definierte lokale Variablen sind nicht initialisiert
  - unabhängig vom Typ
- Wert `null` nicht identisch zu nicht initialisiert
- Best Practice: Weisen Sie einer Referenzvariablen immer `null` zu, wenn kein Objekt zur Hand ist

Bruch  
rational



- Erstellen Sie eine Skizze der Variablen im Speicher für folgenden Quellcode:

```
Tier tier1 = new Tier();  
Tier tier2 = new Tier();  
Tier tier3 = tier1;  
tier2 = tier3;
```

Objektvariablen



- einer Klasse können Variablen zugeordnet werden
  - Objektvariablen, Instanzvariablen, Membervariablen
  - Beispiel: Bestandteile eines Bruchs: Zähler, Nenner
- alle Bestandteile werden in der Klassendefinition aufgelistet:

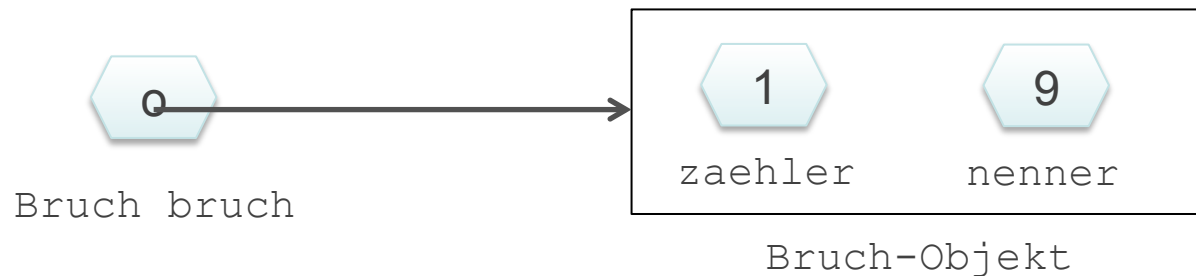
```
class Bruch {  
    int zaehler;  
    int nenner;  
    ...  
}
```

- Anzahl und Typ der Objektvariablen beliebig, im Beispiel:  
 int zaehler; // Objektvariable für den Zähler  
 int nenner; // Objektvariable für den Nenner

- Objektvariablen sind Variablen, ebenso wie bisher verwendete Variablen
- bisher benutzte Variablen = lokale Variablen
- Deklarationssyntax von Objektvariablen und lokalen Variablen ist gleich
- aber: Ort der Deklaration ist entscheidend!
  - Objektvariablen → Elemente von Klassen
  - lokale Variablen → Anweisungen in Methoden
- Benennung von Objektvariablen:
  - wie lokale Variablen (erster Buchstabe klein!)
  - eindeutig innerhalb einer Klasse

- jedes Objekt enthält die Objektvariablen, die in der Klassendefinition festgelegt sind
- Objektvariablen eines Objekts können einzeln angesprochen werden
  - sogenannter Elementzugriff
- Objekt, an das sich ein Elementzugriff richtet: Zielobjekt
- Syntax für den Zugriff auf eine Objektvariable:  
    <Zielobjekt>.<Objektvariablenname>
- <Zielobjekt> ist meist Wert einer Referenzvariablen

- Erzeugen eines Bruch-Objekts mit Wert 1/9:
  - 1. Bruch-Objekt erzeugen und an Referenzvariable zuweisen:  
`Bruch bruch = new Bruch();`
  - 2. Werte für Zähler und Nenner des Zielobjekts einzeln zuweisen:  
`bruch.zaehler = 1;`  
`bruch.nenner = 9;`
  - anschließend: `bruch` Zielobjekt ist mit 1/9 initialisiert



- Elementzugriff spricht Objektvariablen innerhalb eines Objektes an
- gleiche Verwendung wie bei lokalen Variablen!
- Beispiel: Zähler oder Nenner eines Bruch-Objektes
  - in einem Ausdruck verwenden:  
`int i = 5 - bruch.zaehler * 3;`
  - mit Wertzuweisung oder Inkrementoperator modifizieren:  
`bruch.zaehler = 10;`  
`bruch.nenner++;`
  - vergleichen:  
`if ( bruch.zaehler != 0 ){ ...`
- nur die Zugriffssyntax zeigt den Unterschied zwischen Objektvariablen und lokalen Variablen an!

- jedes Objekt hat eigene Objektvariablen
- Elementzugriff richtet sich an eine Objektvariable innerhalb eines Objektes (des Zielobjektes)
- andere Objektvariablen des Zielobjektes und Objektvariablen anderer Objekte bleiben unberührt
- Beispiel

- zwei Brüche erzeugen, mit unterschiedlichen Werten initialisieren:

```
Bruch bruch1 = new Bruch(); // 1/9
```

```
bruch1.zaehler = 1;
```

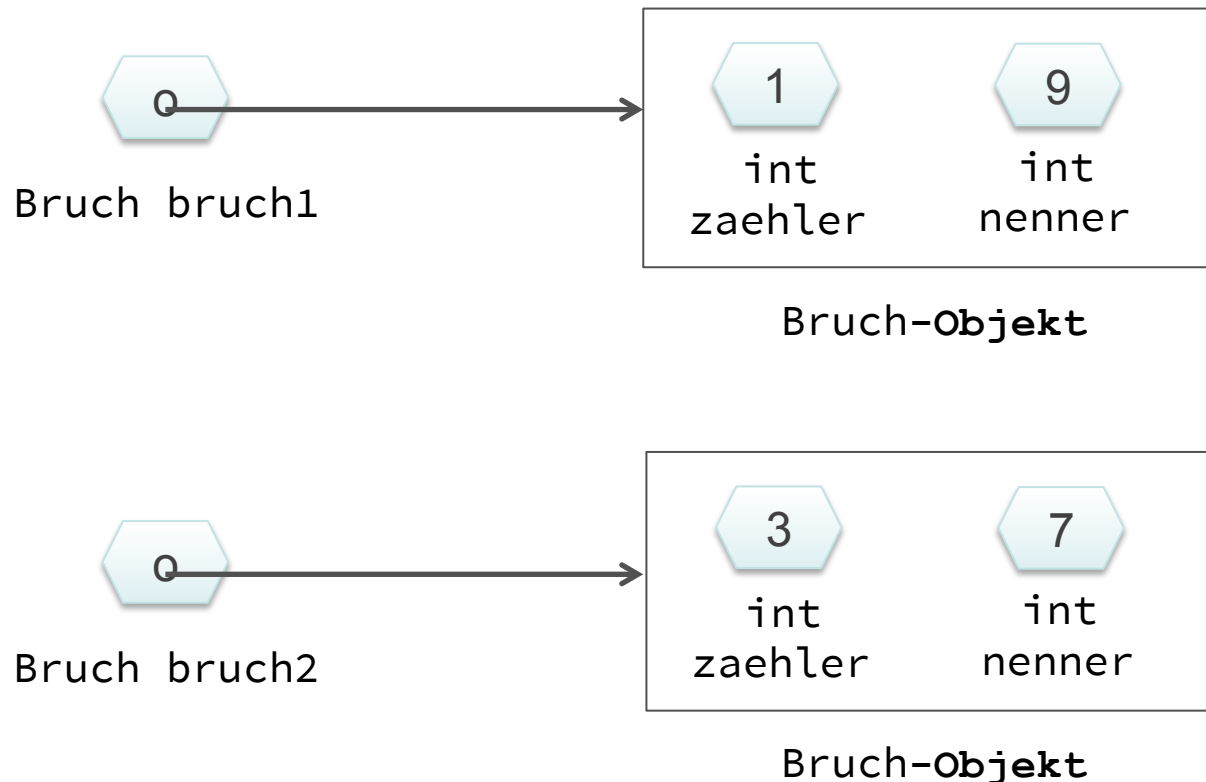
```
bruch1.nenner = 9;
```

```
Bruch bruch2 = new Bruch(); // 3/7
```

```
bruch2.zaehler = 3 * bruch1.zaehler;
```

```
bruch2.nenner = bruch1.nenner - 2;
```

- Beispiel
  - bruch1 und bruch2 sind isolierte, unabhängige Objekte!



- Wertzuweisung **primitiver Typen** kopiert den Wert

```
int a = 1;
```

```
int b = a;
```



int a



int b

- beide Variablen haben den Wert 1:

- Änderungen einer Variablen ist ohne Auswirkungen auf die andere:

```
...
```

```
b++;
```

```
System.out.println(a); // gibt 1 aus
```

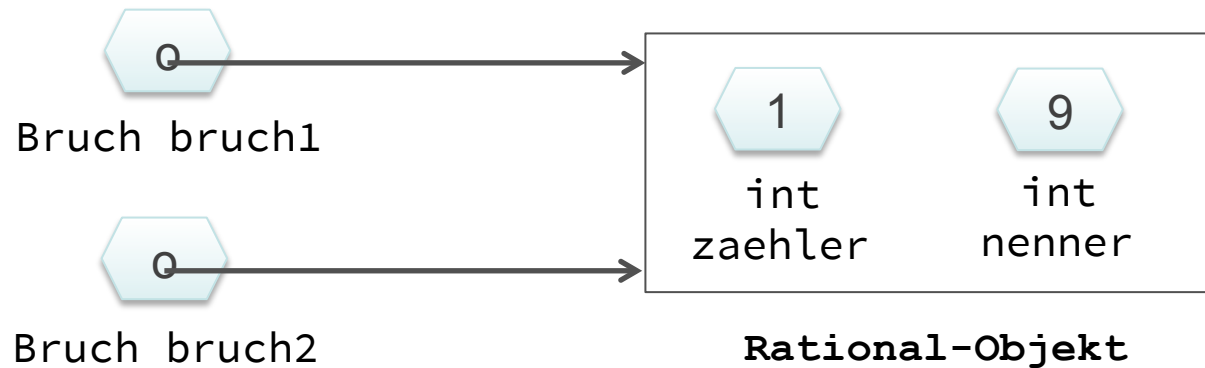
- b inkrementiert auf 2, a immer noch 1.



- Wertzuweisung bei **Referenztypen** kopiert den Zeiger (die Referenz), nicht das Objekt!
- Beispiel:

```
Bruch bruch1 = new Bruch();  
bruch1.zaehler = 1;  
bruch1.nenner = 9;  
Bruch bruch2 = bruch1;
```

  - beide Variablen referenzieren dasselbe Objekt mit Wert 1/9!
  - engl. Bezeichnung, falls jemand mehrere Namen hat: "Aliasing"



→ Änderungen des Objekts sind in beiden Variablen sichtbar:

...

```
bruch2.zaehler++;
```

```
System.out.println(bruch1.zaehler); // gibt 2 aus!
```


- Erweitern Sie die Klasse Tier um eine int-Variable `alter` für das Alter
- Erzeugen Sie ein Tier-Objekt
- Weisen Sie der Variable den Wert 23 zu
- Inkrementieren Sie das Alter um 1.
- Geben Sie das Alter auf der Konsole aus

```
/**  
 * Repräsentation eines Tiers.  
 */  
public class Tier {  
    /**  
     * Programmeinstiegs-Methode.  
     */  
    public static void main(String[] args) {  
        new Tier();  
        new Tier();  
    }  
}
```


Vergleich und Lebensdauer

- Vergleich von Objekten mit `==` prüft die Identität:
  - `true`, wenn beide Operanden ein und dasselbe Objekt sind
  - `false`, wenn die Operanden verschiedene Objekte sind
- Vergleich mit `==` ignoriert den Inhalt der Objekte

```
Bruch bruch1 = ...;  
Bruch bruch2 = bruch1;  
  
if ( bruch1 == bruch2 ){  
    ...
```



```
Bruch bruch1 = new Bruch();  
bruch1.zaehler = 1;  
bruch1.nenner = 9;  
  
Bruch bruch2 = new Bruch();  
bruch2.zaehler = 1;  
bruch2.nenner = 9;  
  
if ( bruch1 == bruch2 ){  
    ...
```



- alle Objektvariablen müssen für die inhaltliche Gleichheit paarweise verglichen werden!

```
Bruch bruch1 = ...;
```

```
Bruch bruch2 = ...;
```

```
if( bruch1.zaehler == bruch1.zaehler &&  
    bruch1.nenner == bruch2.nenner ){  
    ...
```

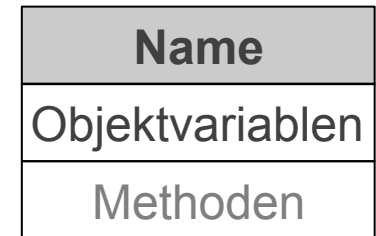
- Standardmethode
  - equals(...)
  - später mehr ...

- lokale Variablen werden ...
  - geschaffen, wenn die Deklaration erreicht wird
  - zerstört, wenn der Block der Deklaration verlassen wird
- Objektvariablen werden ...
  - geschaffen, sobald ein Objekt erzeugt wird (new)
  - zerstört, wenn das Objekt nicht mehr erreichbar ist
    - keine Referenz mehr existiert
    - automatisches "Garbage Collection" (Müll sammeln)
- das heißt:
  - die Lebensdauer von Objekten einschließlich der darin enthaltenen Objektvariablen ist unabhängig von Blockgrenzen!

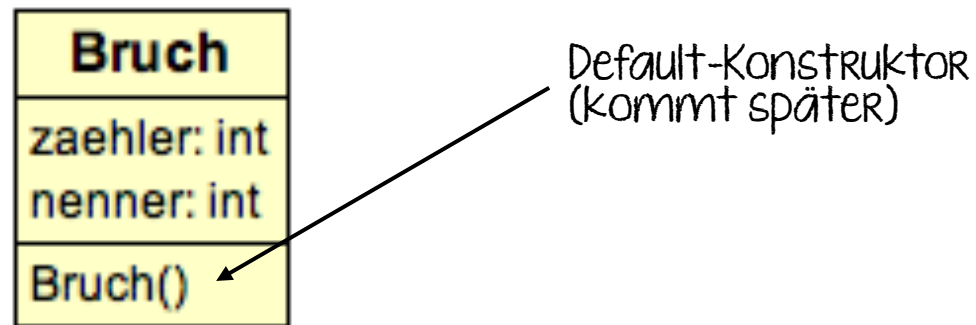
UML



- Klasse = Kasten mit drei „Fächern“
  - Name der Klasse
  - Objektvariablen mit Typ und Name
  - Methoden: später



- Beispiel: Bruch



**Hinweis:** Klassendiagramme können in Eclipse einfach mit dem Plugin *ObjectAid UML Explorer* erzeugt werden (<http://www.objectaid.com/>).

## Aufgabe

- Entwerfen Sie eine Klasse CharDoublePaar
  - die Klasse hat zwei Eigenschaften:
    - ein Zeichen
    - eine Fließkommazahl
- Geben Sie den Quellcode an
- Zeichnen Sie ein UML-Diagramm

- Klassen und Objekte
- Referenztypen
- Objektvariablen
- Vergleich und Lebensdauer
- UML