
# Objektorientierte Programmierung

## Studiengang Medieninformatik

Hans-Werner Lang

Hochschule Flensburg

Vorlesung 5
12.04.2017

# Was bisher geschah...

Hochschule Flensburg
University of Applied Sciences

Objektorientierte Programmierung

- Klassen und Objekte, Attribute und Methoden
- Klassen ableiten, Vererbung von Attributen und Methoden
- Methoden überschreiben und überladen, Polymorphie
- Abstrakte Klasse, abstrakte Methoden, Interfaces

Grafische Benutzungsoberfläche (GUI)

- GUI-Elemente: Frames, Panels, Labels, Buttons, Textfelder
- Reaktion auf Ereignisse mit Listenern

# Klassenattribute

Hochschule Flensburg
University of Applied Sciences

Die Werte von Attributen sind von Objekt zu Objekt unterschiedlich – beispielsweise hat das Fenster *frame1* die Breite 200 und das Fenster *frame2* die Breite 500.

Manchmal gibt es aber auch Attribute, die für alle Objekte einer Klasse gleich sein sollen. Beispielsweise soll für alle erzeugten Objekte unserer Klasse *MyFrame* eine maximale Breite von 1000 zulässig sein. Dies lässt sich durch ein Klassenattribut oder statisches Attribut bewerkstelligen; hierfür wird der Modifizierer `static` verwendet.

```
public class MyFrame extends JFrame
{
    public static int maxwidth=1000;  // Klassenattribut
    ...
    @Override
    public void setSize(int w, int h)
    {
        super.setSize(Math.min(w, maxwidth), h);
    }
}
```

# Klassenattribute

Hochschule Flensburg
University of Applied Sciences

Wenn es uns beliebt, können wir den Wert für die maximale Breite eines Fensters auch ändern:

```
MyFrame.maxwidth=800;
```

Von nun an haben alle neu erzeugten Fenster eine maximale Breite von 800.

Der Zugriff auf Klassenattribute geschieht in ähnlicher Weise wie der Zugriff auf Objektattribute mit der Punkt-Notation.

Statt des Objektnamens wird jedoch der Klassenname angegeben (hier *MyFrame*).

# Klassenkonstanten

Hochschule Flensburg
University of Applied Sciences

Vielleicht ist es nicht sinnvoll, dass die maximale Breite eines Fensters geändert werden kann. Dann deklarieren wir die maximale Breite als Klassenkonstante mithilfe des Modifizierers `final`:

```
public class MyFrame extends JFrame
{
    public static final int MAXWIDTH=1000;  // Klassenkonstante
    ...
```

Anderes Beispiel:

```
public class Bruch
{
    public static final Bruch ZERO=new Bruch(0, 1);
    ...
```

Es ist üblich, Konstanten in Großbuchstaben zu schreiben.

Uns sind bereits Klassenkonstanten begegnet, etwa `Color.GREEN` oder `JFrame.EXIT_ON_CLOSE`.

# Klassenmethoden

Hochschule Flensburg
University of Applied Sciences

Normalerweise werden Methoden auf Objekte angewendet. Beispielsweise ist ein Aufruf der Methode *setVisible* nur dann sinnvoll, wenn er sich auf ein bestimmtes Objekt bezieht, das sichtbar gemacht werden soll:

```
myframe.setVisible(true);
```

Es gibt jedoch auch Methoden, die sich nicht auf ein bestimmtes Objekt beziehen. Eine solche Methode heißt Klassenmethode, eine andere Bezeichnung ist statische Methode. Sie wird durch den Modifizierer `static` gekennzeichnet. Ein Beispiel hierfür kennen wir schon:

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

Die *main*-Methode ist ein Sonderfall, weil sie normalerweise aufgerufen wird, indem wir das Programm starten. Wir können aber auch aus einem Programm heraus eine *main*-Methode einer anderen Klasse aufrufen. Der Aufruf geschieht ähnlich wie beim Zugriff auf Klassenattribute durch Angabe des Klassennamens, hier *Main1*:

```
Main1.main(null);
```

# Hilfsfunktionen in Klassen



Hilfsfunktionen, die wir in Klassen benötigen, kennzeichnen wir mit `private static`, beispielsweise in unserer Klasse *Bruch*:

```
private static int ggt(int a, int b)
```

Denn Hilfsfunktionen sind nur in der betreffenden Klasse relevant, daher `private`, und sie beziehen sich nicht auf ein bestimmtes Objekt, daher `static`.

Klassenattribute, Klassenkonstanten und Klassenmethoden werden in Eclipse in Kursivschrift dargestellt.

# Factory-Methode

Hochschule Flensburg
University of Applied Sciences

Eine Klassenmethode, die ein Objekt der Klasse erzeugt, heißt Factory-Methode.

Beispiel (anstelle von Klassenkonstante *ZERO*):

```
public static Bruch zero()
{
    return new Bruch(0, 1);
}
```

Aufruf z.B. folgendermaßen:

```
Bruch a=new Bruch(1, 2);
boolean g=a.isGreater(Bruch.zero());
```

Hochschule Flensburg
University of Applied Sciences

Wir haben nun die wesentlichen Eigenschaften und Möglichkeiten der objekt-orientierten Programmierung kennengelernt. Es fehlen noch drei wichtige Details:

## 1. Initialisierung von Attributen

```
public class Circle
{
    // Attribute
    public int left, top, diameter;
    public Color fill;

    ...
}
```

Attribute werden bereits bei der Deklaration mit einem Wert initialisiert. Bei Attributen vom Typ `int` ist dies der Wert 0, bei `double` der Wert 0.0, bei `boolean` der Wert `false`.

Attribute, die auf ein Objekt verweisen, werden mit dem speziellen Wert `null` initialisiert (hier: *fill*)

# Initialisierung von Attributen

```
public class Circle
{
    // Attribute
    public int left, top, diameter=10;
    public Color fill=Color.BLACK;

    ...
}
```

Attribute lassen sich bereits bei der Deklaration mit gewünschten Default-Werten initialisieren. Jedes Objekt, das neu erzeugt wird, besitzt dann diese Attribut-Werte – es sei denn, im Konstruktor werden andere Werte für die Attribute festgelegt.

Natürlich lassen sich in entsprechenden Methoden die Attribut-Werte auch nach Erzeugung des Objekts noch verändern, etwa in der Methode *setFill* der Klasse *Circle*:

```
public void setFill(Color c)
{
    this.fill=c;
}
```

# 2. Das `null`-Objekt



```
Bruch a, b, c;
a=new Bruch(1, 2);
b=new Bruch(3, 4);
```

Welchen Wert hat c ? Solange Objekt-Variablen kein Objekt zugewiesen wird, z.B. mit `new`, verweisen sie auf kein Objekt. Dieses nicht vorhandene Objekt ist das `null`-Objekt.

Oder korrekter ausgedrückt: Die Variable *c* enthält einen `null`-Verweis. Beim Versuch, eine Methode von *c* aufzurufen, etwa `c.add(a)`, wird der Laufzeitfehler *NullPointerException* ausgelöst.

Wir können prüfen, ob eine Objekt-Variable den Wert `null` enthält:

```
if (c!=null)
    ...
```

# 3. Umhüllungs-Klassen (*Wrapper*-Klassen)

Hochschule Flensburg
University of Applied Sciences

Java ist keine rein objektorientierte Sprache, denn es gibt die elementaren Datentypen `int`, `double` und `boolean`. Entsprechende Werte dieser elementaren Datentypen sind keine Objekte. Der Vorteil besteht darin, dass sich die Werte einfacher speichern lassen: Anstatt einen Verweis auf ein Objekt zu speichern, wird der Wert direkt gespeichert.

Der Nachteil besteht darin, dass sich Werte dieser elementaren Datentypen nicht wie Objekte behandeln lassen: Sie sind nicht von der Oberklasse *Object* abgeleitet, und sie haben keine Attribute und Methoden.

Abhilfe bieten die Umhüllungs-Klassen, die für jeden elementaren Datentyp vorhanden sind:

`Integer` für `int`
`Double` für `double`
`Boolean` für `boolean`

Das Schöne ist: Die Werte werden automatisch ineinander umgewandelt:

```
Integer i=3;
int j=i;
```

# Attribute und Methoden von Umhüllungs-Klassen

Hochschule Flensburg
University of Applied Sciences

Wichtige Attribute und Methoden der Umhüllungs-Klasse *Integer*:

```
Integer.MAX_VALUE;    // Klassenkonstante: 2147483647
Integer.MIN_VALUE;    // Klassenkonstante: -2147483648

// Klassenmethode:
int n=Integer.parseInt(txt1.getText());  // String -> Integer

// Integer implementiert das Interface Comparable
Integer i=3, j=4;
int v=i.compareTo(j); // Methode compareTo aus Interface Comparable
```

Attribute der Umhüllungs-Klassen *Double*:

```
Double.POSITIVE_INFINITY;    // unendlich
Double.NEGATIVE_INFINITY;    // - unendlich
```

# Zusammenfassung

Hochschule Flensburg
University of Applied Sciences

- Klassenattribute (`static`) , Klassenkonstanten (`static final`)
- Klassenmethoden (`static`)
- Attribute initialisieren: `public int x=10, y=20;`
- Das `null`-Objekt: `public Color fill=null;`
- Umhüllungsklassen (*Wrapper*-Klassen): `Integer`, `Double`, `Boolean`
- Attribute und Methoden der Umhüllungsklassen

Beim nächsten Mal:

- *Refactoring*: Ein fertiges, funktionierendes Programm verbessern
- Programme testen mit Unit-Tests