Hochschule Flensburg
University of Applied Sciences

Objektorientierte Programmierung

Studiengang Medieninformatik

Hans-Werner Lang

Hochschule Flensburg

Vorlesung 2
22.03.2017

Hochschule
Flensburg
University of
Applied Sciences

Was bisher geschah...

- Klassen und Objekte
- Attribute und Methoden
- Klasse *Bruch*
- Java-Klasse *JFrame*

Klassen erweitern

Hochschule Flensburg
University of Applied Sciences

John B. Dunlop hat das Rad nicht neu erfunden, sondern es nur modifiziert und es um einen luftgefüllten Reifen erweitert.

Wir können ebenso eine vorhandene Klasse erweitern – um weitere Attribute und um zusätzliche Methoden – oder sie in anderer Weise modifizieren. Vorhandene Attribute und Methoden bleiben erhalten.

```
public class MyFrame extends JFrame
{
    // Konstruktor
    public MyFrame()
    {
        super(); // Aufruf des Konstruktors der Basisklasse (superclass)
        setVisible(true);     // Objekte unserer neuen Klasse
    }                         // sind sofort sichtbar
}
```

Der Standardkonstruktor der Basisklasse *JFrame* wird automatisch aufgerufen, der Aufruf `super();` kann also weggelassen werden.

Vererbung

Die Methoden der Basisklasse *JFrame* sind auf Objekte der abgeleiteten Klasse *MyFrame* anwendbar. *)

```
MyFrame myframe=new MyFrame();
myframe.setSize(600, 400);
myframe.setTitle("This is MyFrame");
```

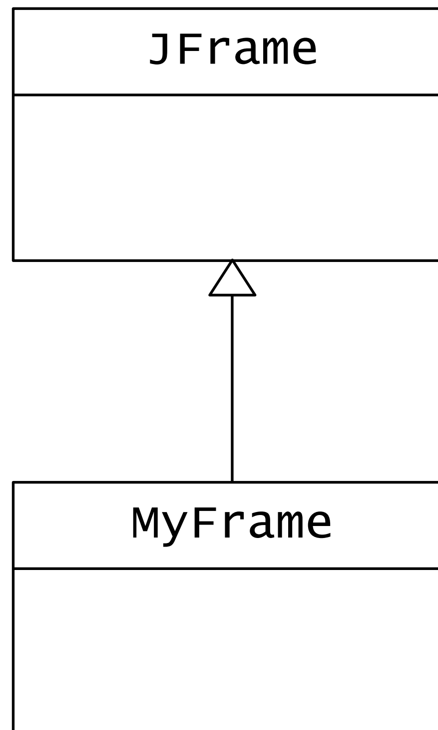
Die Attribute der Basisklasse sind in der abgeleiteten Klasse ebenso vorhanden. *)

Dies wird als <u>Vererbung</u> bezeichnet. Die abgeleitete Klasse <u>erbt</u> die Attribute und Methoden der Basisklasse. *)

*) außer wenn sie in der Basisklasse als `private` deklariert sind

Klassendiagramm

Hochschule Flensburg
University of Applied Sciences

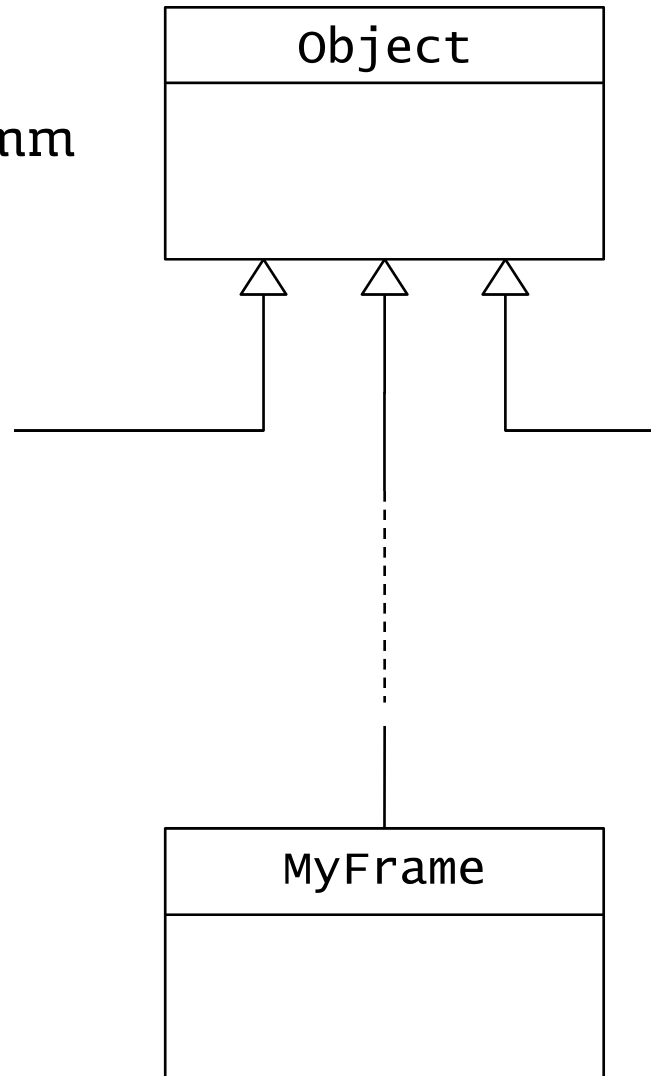


Basisklasse
(Oberklasse,
superclass)

abgeleitete Klasse
(Unterklasse,
subclass)

Klassendiagramm

Hochschule Flensburg
University of Applied Sciences



alle Klassen sind automatisch von einer gemeinsamen Oberklasse *Object* abgeleitet

Methoden überschreiben

Wir implementieren eine Methode, die in der Basisklasse vorhanden ist, neu.

```
public class Bruch // (extends Object)
{
    int zaehler, nenner;
    ...
    @Override   // Methode der Klasse Object überschreiben
    public String toString()
    {
        return zaehler+"/"+nenner;
    }
}
```

Die Methode *toString* wird automatisch beim Aufruf von *print* ausgeführt:

```
System.out.println(new Bruch(2, 3));     // Ausgabe: 2/3
```

Methoden überschreiben

Wir implementieren eine Methode, die in der Basisklasse vorhanden ist, neu.

```
public class MyFrame extends JFrame
{
    ...

    @Override   // überschreiben
    public void setTitle(String s)
    {
        super.setTitle("--- "+s+" ---");
    }
}
```

Mit `super` lässt sich weiterhin auf Methoden der Basisklasse (*superclass*) und damit auf die ursprüngliche Methode *setTitle* zugreifen.

Methoden überladen

Hochschule Flensburg University of Applied Sciences

Wir implementieren zusätzliche Versionen von vorhandenen Methoden:

```
// Methode setSize der Basisklasse überladen (nur 1 Parameter),
// festes Seitenverhältnis des Fensters
public void setSize(int w)
{
    super.setSize(w, (int)(w*0.618));
}

// Methode setVisible der Basisklasse überladen (kein Parameter)
public void setVisible()
{
    ? ? ?
}
```

Methoden überladen

Wir implementieren zusätzliche Versionen von vorhandenen Methoden:

```
// Methode setSize der Basisklasse überladen (nur 1 Parameter),
// festes Seitenverhältnis des Fensters
public void setSize(int w)
{
    super.setSize(w, (int)(w*0.618));
}

// Methode setVisible der Basisklasse überladen (kein Parameter)
public void setVisible()
{
    super.setVisible(true);
}
```

Konstruktoren überladen

Wir implementieren zusätzliche Konstruktoren einer Klasse:

```
// vorhandener Konstruktor
public MyFrame()
{
     setVisible();
     setDefaultCloseOperation(EXIT_ON_CLOSE);
}

// weiterer Konstruktor
public MyFrame(Color c)
{
    this();    // Aufruf des vorhandenen Konstruktors
    getContentPane().setBackground(c);
}
```

Schlüsselwort `this`

Methoden werden auf Objekte angewendet, z.B. die Methode *mul* auf das Objekt a:

```
c=a.mul(b);
```

Wie aber greifen wir in der Definition der Methode *mul* auf das Objekt a zu?

Das Objekt, auf das die Methode angewendet wird, lässt sich mit `this` ansprechen.
Wir erinnern uns:

```
public Bruch mul(Bruch other)
{
    int z=this.zaehler*other.zaehler;
    int n=this.nenner*other.nenner;
    return new Bruch(z, n);
}
```

Schlüsselwort `this`

Das Schlüsselwort `this` kann weggelassen werden.

Eine Ausnahme ist der Aufruf eines anderen Konstruktors der Klasse in der Definition eines Konstruktors.

Beispielsweise wollen wir Brüche mit dem Nenner 1, also ganze Zahlen, folgendermaßen erzeugen, also ohne den Nenner 1 anzugeben.

```
Bruch a=new Bruch(3);  // statt new Bruch(3, 1)
```

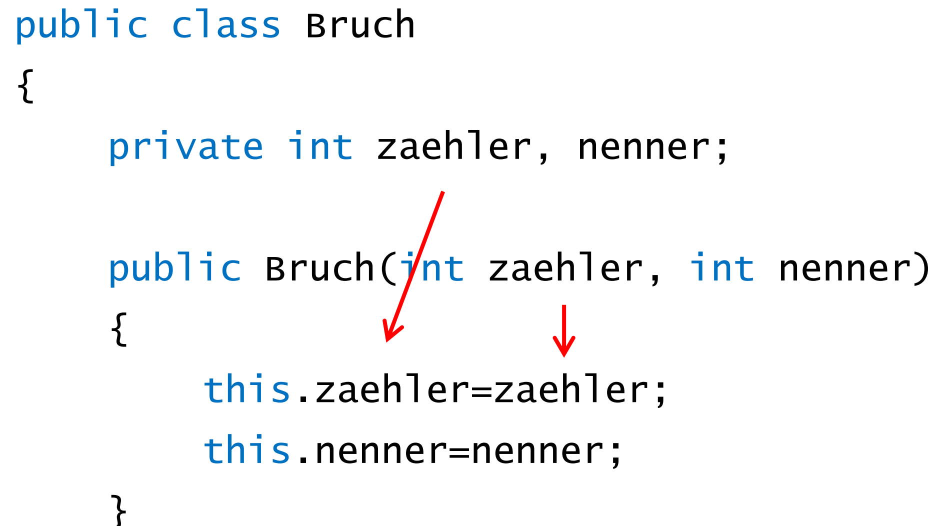
Wir greifen bei der Definition des entsprechenden Konstruktors auf den anderen Konstruktor zurück.

```
public Bruch(int z)
{
    this(z, 1);
}
```

Hochschule Flensburg
University of Applied Sciences

Schlüsselwort this

Das Schlüsselwort this wird ebenfalls gebraucht, um zwischen namensgleichen Attributen und Parametern zu unterscheiden:



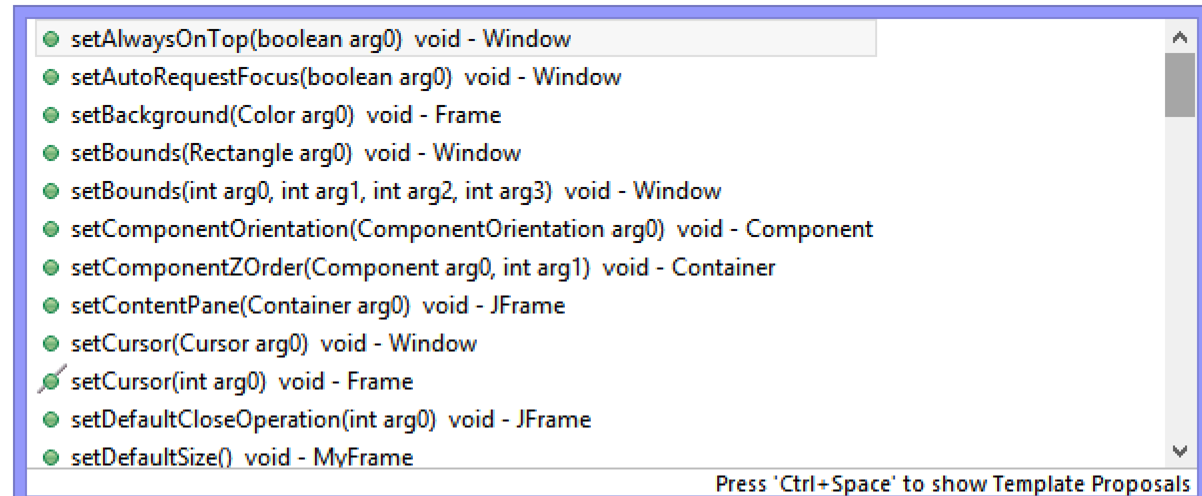
```
public class Bruch
{
    private int zaehler, nenner;

    public Bruch(int zaehler, int nenner)
    {
        this.zaehler=zaehler;
        this.nenner=nenner;
    }
```

Schlüsselwort this

Das Schlüsselwort this ist außerdem hilfreich, um von Eclipse die zur Verfügung stehenden Methoden der aktuellen Klasse anzeigen zu lassen:

```
public class MyFrame extends JFrame
{
    // Konstruktor
    public MyFrame()
    {
        this.s
    }
}
```



Hochschule Flensburg University of Applied Sciences

Zusammenfassung

- Eine Klasse kann von einer anderen Klasse abgeleitet werden:

```
public class MyFrame extends JFrame
```

- Attribute und Methoden werden von der Oberklasse geerbt, Konstruktoren nicht
- Methoden können überschrieben werden, auf entsprechende Methoden der Oberklasse kann mit `super.`*methodenname* zugegriffen werden
- Im Konstruktor der abgeleiteten Klasse kann mit `super(…)` ein Konstruktor der Oberklasse (*superclass*) aufgerufen werden
- Fehlt ein Aufruf von `super(…)` oder `this(…)` im Konstruktor der abgeleiteten Klasse, wird automatisch der Standardkonstruktor der Oberklasse aufgerufen (`super()`)
- Aufruf von `super(…)` oder `this(…)` nur als erste Anweisung des Konstruktors

Hochschule Flensburg
University of Applied Sciences

Datenkapselung

Vom Innenleben einer Klasse ist nur das Notwendige nach außen sichtbar (*information hiding*). Durch die Kapselung werden nur Informationen über das Verhalten einer Klasse nach außen sichtbar, nicht aber die Implementierung.

Für Attribute und Methoden lässt sich die Sichtbarkeit durch die folgenden Schlüsselwörter festlegen:

Sichtbarkeitsarten

`public` (+)	aus allen anderen Klassen heraus zugreifbar,
`private` (-)	nur innerhalb der eigenen Klasse zugreifbar,
`protected` (#)	nur aus der eigenen Klasse und aus davon abgeleiteten Klassen zugreifbar,
package (~)	(wenn nichts angegeben) nur aus Klassen des eigenen Pakets zugreifbar.

Hochschule Flensburg
University of Applied Sciences

Beim nächsten Mal:

- Abstrakte Klasse
- Interface