



# Vererbung zur Vermeidung von Coderedundanzen

Buch, Kapitel 10; Verständliche, wartbare und wiederverwendbare Klassen

J. Kleimann, H.-W. Sehring, D. Versick, F. Zimmermann

Studiengang Wirtschaftsinformatik

# Vorlesungsinhalt

- 1 Lerninhalte
- 2 Das Netzwerkbeispiel
- 3 Vererbung
- 4 Vererbung und Exemplarvariablen
- 5 Vererbung und Konstruktoren
- 6 Vererbung und Methoden
- 7 Subtypen und Class Casts
- 8 Zusammenfassung

## ENTEPRISE ARCHITECTURE MADE EASY

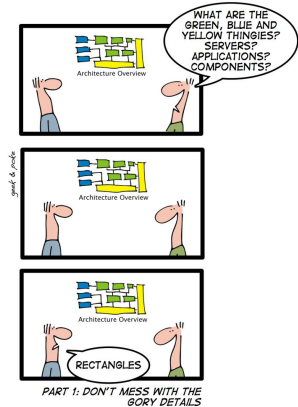


Abbildung: CC-BY-3.0 Oliver Widder

# Lernziele

In dieser Lektion<sup>1</sup> werden Sie

- Vererbung im Zusammenhang mit
  - Exemplarvariablen,
  - Konstruktoren und
  - Exemplarmethoden

kennenlernen.

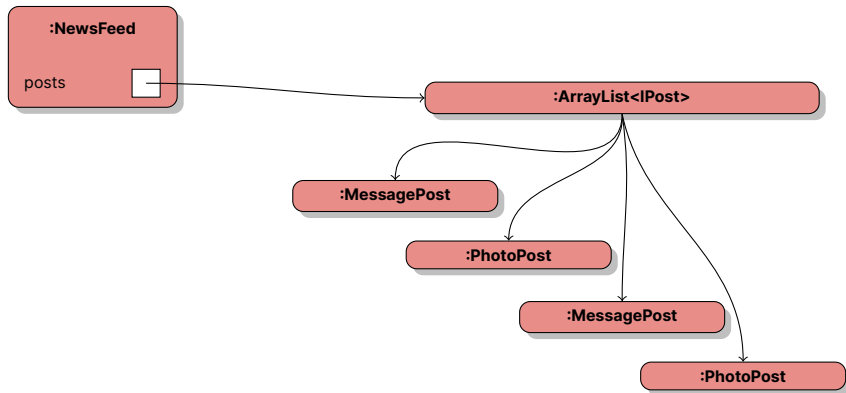
---

<sup>1</sup>basiert auf Folien und Beispielen von Barnes, Kölling

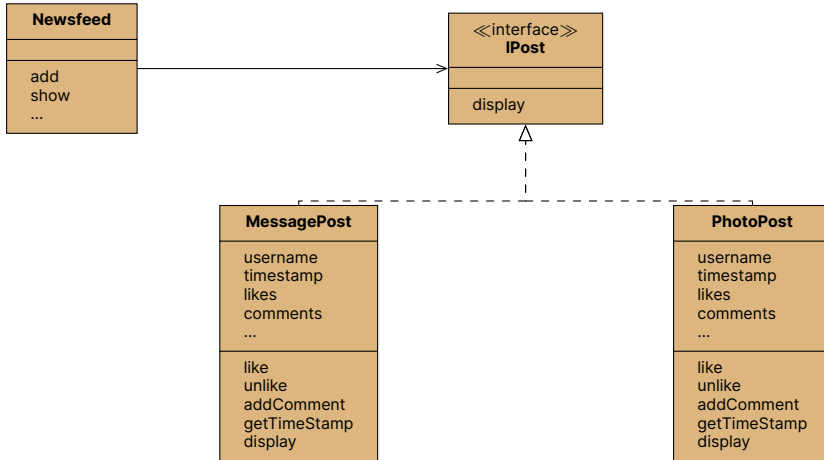
# Das Netzwerkbeispiel

- Wir werden in dieser Lektion weiter am Netzwerkbeispiel arbeiten.
- Ziel ist es Code-Duplikate innerhalb einer Klassenhierarchie zu vermeiden.
- Wir werden die Vererbung von Exemplarvariablen und -methoden nutzen.

# Netzwerkobjektmodell nach der letzten Lektion



# Ähnlichkeiten in den Netzwerkklassen



# MessagePost Quellcode (ein Ausschnitt)

```
public class MessagePost implements
    IPost{
    private String username;
    private String message;
    private long timestamp;
    private int likes;
    private ArrayList<String> comments;

    public MessagePost(String author ,
        String text) {
        username = author;
        message = text;
```

```
        timestamp = System.
            currentTimeMillis();
        likes = 0;
        comments = new ArrayList<>();
    }

    public void addComment(String text)
        ...
    public void like() ...
    public void display() ...
        ...
    }
```

# PhotoPost SourceCode (ein Ausschnitt)

```
public class PhotoPost implements
    IPost{
    private String username;
    private String filename;
    private String caption;
    private long timestamp;
    private int likes;
    private ArrayList<String> comments;

    public PhotoPost(String author,
        String filename,
            String caption) {
        username = author;
```

```
        this.filename = filename;
        this.caption = caption;
        timestamp = System.
            currentTimeMillis();
        likes = 0;
        comments = new ArrayList<>();
    }

    public void addComment(String text)
        ...
    public void like() ...
    public void display() ...
        ...
    }
```



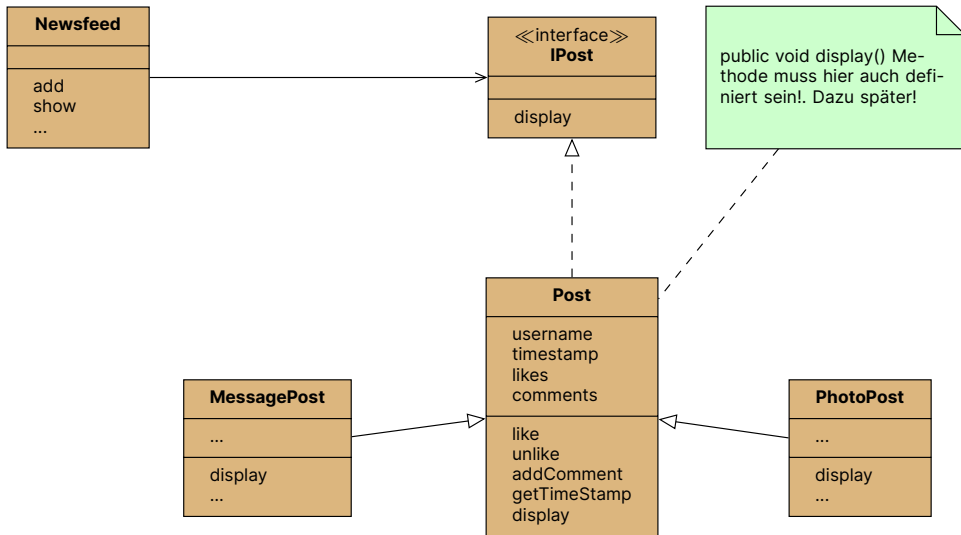
# Immer noch Kritik am Netzwerk

Duplizierter Quellcode:

- Die MessagePost und PhotoPost sind in vielen Methoden identisch. Ausnahme display() hier ist einiges verschieden.
- Dies macht Wartung **schwieriger** und arbeitsaufwändiger.
- Erhöhte Gefahr von Fehlern durch nicht korrekte Wartung.

```
// gemeinsame Exemplarvariablen  
private String username;  
private long timestamp;  
private int likes;  
private ArrayList<String> comments;  
  
//gemeinsame Methoden  
public void addComment(String text) ...  
public void like() ...  
public void display() ...  
...
```

# Unter Verwendung von Vererbung



# Vererbung

- Definieren einer **Oberklasse**: Post
- Definieren von **Unterklassen**: MessagePost und PhotoPost
- Die Oberklasse (**Superklasse**) definiert gemeinsame Exemplarvariablen (auch bekannt als Instanzvariablen, Attribute oder Felder)
- Die Oberklasse definiert gemeinsame Exemplarmethoden, die die gemeinsamen Exemplarvariablen bearbeiten.
- Die Unterklassen (**Subklasse**) **erben** die Exemplarvariablen von der Oberklasse. Das bedeutet, das Objekte der Unterklassen diese Exemplarvariablen auch haben. Sie erben auch die Exemplarmethoden der Oberklasse.
- Die Unterklassen fügen bei Bedarf weitere Exemplarvariablen oder -methoden hinzu.

# Synonyme

- Statt Oberklasse verwendet man auch den Begriff Generalisierung.
- Statt Oberklasse verwendet man auch den Begriff Supertyp.
- Statt Unterklasse verwendet man auch den Begriff Spezialisierung.
- Statt Unterklasse verwendet man auch den Begriff Subtyp.

# Vererbung in Java

- Das Schlüsselwort `extends` signalisiert eine Vererbungsbeziehung.
- Nicht **private** Exemplarvariablen und -methoden werden von der Oberklasse an die Unterklasse vererbt.

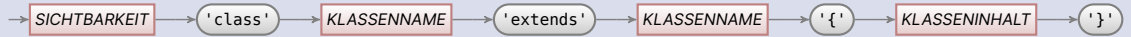
```
public class Post implements IPost{  
    ...  
}
```

```
public class PhotoPost extends Post {  
    ...  
}
```

```
public class MessagePost extends Post {  
    ...  
}
```

# Klassendefinition, Vererbung (Syntax)

Verallgemeinert (Klassendefinition, Vererbung)



# Oberklasse

```
public class Post implements IPost{  
    private String username;  
    private long timestamp;  
    private int likes;  
    private ArrayList<String> comments;  
  
    // Konstruktor ausgelassen  
  
    public void addComment(String text) {...}  
    public void like () {...}  
    public void unlike () {...}  
    public long getTimestamp () {...}  
    public void display (){}  
}
```

# Unterklassen erweitern die Oberklasse

```
public class MessagePost extends Post {  
    private String message;  
  
    // Konstruktor ausgelassen  
  
    public void display() { ... }  
    ...  
}
```

```
public class PhotoPost extends Post {  
    private String filename;  
    private String caption;  
  
    // Konstruktor ausgelassen  
  
    public void display() { ... }  
    ...  
}
```



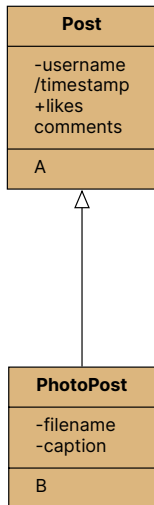
# Drei Aspekte der Vererbung

- Die Erklärung der Mechanik der Vererbung erfordert die Betrachtung der folgenden drei Aspekte:
  - Exemplarvariablen
  - Konstruktoren zum Initialisieren der Exemplarvariablen
  - Exemplarmethoden

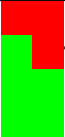

# Vererbung von Exemplarvariablen

- Alle Objekte der Unterklasse speichern auch die Exemplarvariablen der Oberklasse.
- **Aber:** Es gibt auch ein Geheimnisprinzip zwischen Ober- und Unterklasse. Da die Oberklasse nicht weiß welche Unterklassen es gibt, soll durch das Geheimnisprinzip verhindert werden, dass die Variablen der Oberklasse durch Unterklassen in einen inkonsistenten Zustand gebracht werden.
- Es ist schlechter Programmierstil in der Unterklasse auf die Exemplarvariablen der Oberklasse zuzugreifen. Dazu soll man die Exemplarmethoden der Oberklasse zu verwenden.
- Java unterstützt das Geheimnisprinzip: **private** Exemplarvariablen können nicht in der Unterklasse benutzt werden.
- Der Sichtbarkeitsmodifikator **protected** für Variablen der Oberklasse erlaubt den Zugriff auf die Variable in den Unterklassen.
- Die Nutzung von **protected** soll (für Exemplarvariable) vermieden werden.

# Vererbung von Exemplarvariablen



Zugriff in der Klasse PhotoPost auf Exemplarvariablen

Urpung	Deklaration	Speicher	Zugriff
Post	<code>private String username;</code>	muster	 <sup>a</sup>
	<code>List&lt;String&gt; comments;</code>	null	
	<code>protected long timestamp;</code>	42424242	
	<code>public int likes;</code>	10	
PhotoPost	<code>private String filename;</code>	pic.jpg	
	<code>private String caption;</code>	Sun rise	

Die Exemplarmethoden A (Klasse Post) arbeiten nur mit Exemplarvariablen der Klasse Post. Die Exemplarmethoden B (Klasse PhotoPost) können mit allen grünen Exemplarvariablen arbeiten.

<sup>a</sup> „package private“, Zugriff wenn beide Klassen in demselben Paket

# Überdecken von Variablen

- Da man auf **private** Exemplarvariablen der Oberklasse nicht zugreifen kann, zeigt der Compiler einen Fehler, wenn man den Zugriff versucht.
- Anfänger legen dann gerne eine Exemplarvariable gleichen Namens in der Unterklasse an, und freuen sich dass der Compile-Fehler verschwindet.
- Aber: jetzt gibt es zwei Exemplarvariablen desselben Namens, eine in der Oberklasse, eine in der Unterklasse. Man sagt, dass die Variable der Unterklasse die Variable der Oberklasse **überdeckt**.
- Da die zwei Variablen unterschiedliche Werte haben können, funktionieren die Klassen nicht so, wie man es sich vorstellt. Ein schwer (für Anfänger unmöglich) zu findender Fehler.

# Beispielcode

```
public class Post
    implements IPost{
    protected String username;

    ...
}
```

```
public class MessagePost
    extends Post{
    private String message;

    public void display(){
        ...username...;
        ...message...;
    }
}
```

# Beispielerklärung

- In der `display()`-Methode von `MessagePost`-Exemplaren soll der Nachrichtentext (`message` aus der Klasse `MessagePost`) **und** der Nutzernamen (`username` aus der Klasse `Post`) ausgegeben werden.
- Der Zugriff auf `message` in der `display()` Methode ist problemlos, selbst wenn `message` **private** deklariert wird.
- Der Zugriff auf `username` (in der Klasse `Post`) in `display()` (in der Klasse `MessagePost`) funktioniert nur, wenn `username` als **protected** deklariert wird. Ein **private** deklariertes `username` erzeugt einen Compile-Fehler in `MessagePost`.

# Umgang mit dem Problem `display`

- Die Nutzung von **protected** soll vermieden werden, weil es eine Verletzung des Geheimnisprinzips darstellt.
- Uns fehlen aber noch Kenntnisse, die erst später vermittelt werden.
- **protected** getter- und setter-Methoden sind auch keine richtige Lösung.
- In dieser Lektion werden wir deshalb die klassenspezifischen Exemplarvariablen aus den `display` Methoden entfernen und nur die Exemplarvariablen aus `Post` verwenden. Damit kann `display` wie die anderen Methoden in `Post` programmiert werden.

# Konstruktoren

- Konstruktoren initialisieren die Objekte einer Klasse.
- Die Exemplarvariablen der Unterklassen werden in den Unterklassen initialisiert
- Die Exemplarvariablen der Oberklassen werden in den Oberklassen initialisiert
- Konstruktoren der Unterklassen müssen die **Konstruktoren der Oberklasse** aufrufen.



# Vererbung und Konstruktoren

```
public class Post implements IPost{  
    private String username;  
    private long timestamp;  
    private int likes;  
    private ArrayList<String> comments;  
  
    /**  
     * Initialise the fields of the  
     * post.  
     */  
}
```

```
public Post(String author) {  
    username = author;  
    timestamp = System.  
        currentTimeMillis();  
    likes = 0;  
    comments = new ArrayList<>();  
}  
  
    // Methoden ausgelassen  
}
```

# Vererbung und Konstruktoren

```
public class MessagePost extends Post {  
    private String message;  
  
    /**  
     * Constructor for objects of class MessagePost  
     */  
    public MessagePost(String author, String text) {  
        super(author);  
        message = text;  
    }  
  
    // Methoden ausgelassen  
}
```

# Aufruf von Konstruktoren der Oberklasse

- Die Konstruktoren von Unterklassen müssen einen „super“ Aufruf beinhalten.
- Wenn keiner angegeben wird, dann fügt der Übersetzer (Compiler) einen Aufruf eines Konstruktors der Oberklasse ohne Parameter ein. Dies entspräche einem Aufruf von **super()**.
  - die Übersetzung schlägt fehl falls die Oberklasse keinen Konstruktor ohne Parameter beinhaltet
- Ein solcher Aufruf muss die erste Anweisung in den Konstruktoren der Unterklasse sein.
- Es darf nur einen Aufruf eines Konstruktors der Oberklasse geben.

Nebenbei bemerkt: es gibt auch einen **this()** Aufruf.

# Aufruf von Konstruktoren der Oberklasse

```
public class Post {  
    /** Constructor for objects of  
        class Post */  
    public Post(String author) {  
        this.author = author;  
    }  
    // Methoden ausgelassen  
}
```

```
public class Post extends Object {  
    /** Constructor for objects of  
        class Post */  
    public Post(String author) {  
        super();  
        this.author = author;  
    }  
    // Methoden ausgelassen  
}
```

Der Java-Compiler fügt die auf der rechten Seite gelb hervorgehobenen Elemente ein, da

- Jede Klasse in Java von der Klasse `Object` erbt (oder von einer Unterklasse von `Object`, die dann aber angegeben werden muss)
- Der Aufruf des default Konstruktors obligatorisch ist (am Ende auch immer des default Konstruktors der Klasse `Object`).

# Übung siehe Moodle Test Foliensatz 9

1. Führen Sie in das Projekt eine Klasse `Post` ein. Siehe dazu Folie 15. Verwenden sie zunächst als Sichtbarkeit der Attribute **protected**<sup>2</sup> und nicht wie auf der Folie **private**. Lassen Sie die Methoden weg.
2. Fügen sie in `Post` einen Konstruktor hinzu. Siehe dazu Folie 25.
3. `MessagePost` und `PhotoPost` erben von `Post`. Siehe Folie 16.
4. Löschen Sie die Attribute in den Unterklassen, die in `Post` definiert werden.
5. Fügen Sie Konstruktoren in den Unterklassen `MessagePost` und `PhotoPost` ein. Siehe Folie 26.
6. Um `Post` compilieren zu können, fügen Sie in `Post` noch eine Exemplarmethode **public** `void display() {}` ein. Wir werden uns in der nächsten Übung von dieser Methode wieder trennen.
7. Testen Sie die Anwendung.

---

<sup>2</sup>eigentlich schlecht, etwas besser sind `protected` getter Methoden

# Methodenvererbung

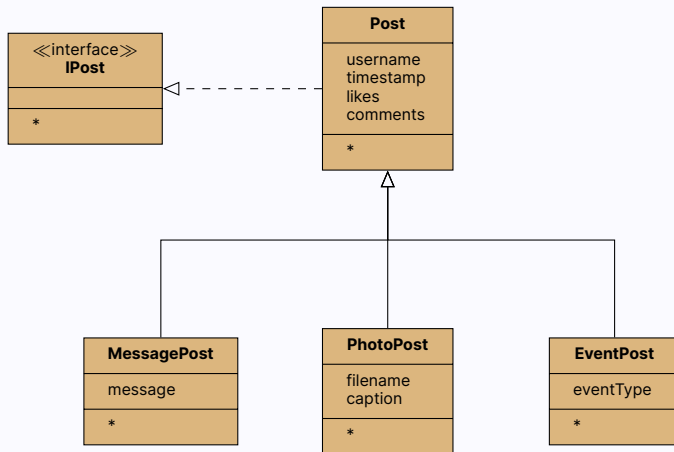
Nicht nur Exemplarvariablen sondern auch Exemplarmethoden können vererbt werden.

- Hat eine Oberklasse eine Methode, kann diese Methode auch auf Objekten der Unterklassen ausgeführt werden.
- Methoden der Oberklasse greifen allerdings nur auf Exemplarvariablen der Oberklasse zu.
- Es gibt den Sichtbarkeitsmodifizier **protected** auch für Exemplarmethoden.

# Übung 2

1. Nun da die Attribute umgezogen sind, wird es Zeit, auch die entsprechenden Methoden zu verschieben.
2. Kopieren Sie die Methoden, die nur Variablen aus `Post` nutzen aus `MessagePost` und `PhotoPost` in die Klasse `Post`, und löschen die Methoden aus den Unterklassen.
3. Achtung: Bei der `display` Methode haben Sie das Problem, dass diese Methode in den beiden Unterklassen anders ist, weil auf die Exemplarvariablen der Unterklassen zugegriffen wird. Entfernen Sie diese Zugriffe deshalb und ziehen Sie die modifizierte `display` Methode in die Klasse `Post` hoch, wie die anderen Methoden.

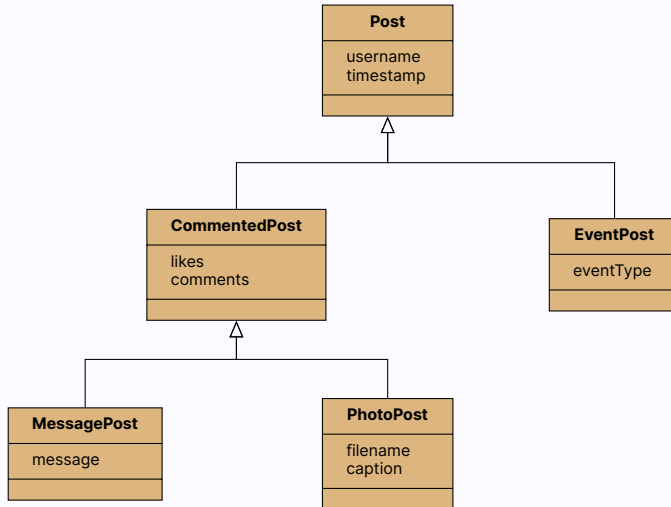
# Erweiterung durch neue Klassen



\*Methoden ausgeblendet



# Tiefere Hierarchien

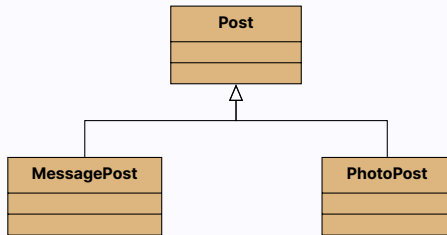


# Klassenhierarchie

- Klassen definieren Typen
- Unterklassen definieren **Subtypen**
- Exemplare von Subtypen können überall dort verwendet werden wo Exemplare von Supertypen erforderlich sind. (Dies wird auch als **Substitution** bezeichnet.)
- Eine Klassenhierarchie ist eine „ist ein“ (isA) Hierarchie.
- Jeder PhotoPost ist ein Post. Jeder Post ist ein IPost.
- Nicht jeder Post ist ein PhotoPost.
- Nicht jeder IPost ist ein Post. Diese Aussage ist selbst dann richtig, wenn es nur eine Klasse Post gibt, die IPost implementiert. Java erlaubt es über verschiedene Mechanismen (anonyme innere Klassen oder Reflection) IPost-Exemplare zu erstellen, die nicht Post-Exemplare sind.

# Subtypen und Zuweisungen

## Variablen und Subtypen



Instanzen von Unterklassen können zu Variablen vom Typ der Oberklasse zugewiesen werden

```
Post p1 = new Post();
Post p2 = new MessagePost();
Post p3 = new PhotoPost();
```

# Zuweisungskompatibilität

- Wir könnten Subtypen zu Supertypen zuweisen ...
- ... aber wir können nicht Supertypen zu Subtypen zuweisen

```
Post p;  
MessagePost m = new MessagePost();  
p = m; // correct ⇒ Widening Reference Conversion  
m = p; // compile-time error! ⇒ Narrowing Reference Conversion
```

# Typkonvertierung (Casting)

Explizite Typkonvertierung (Casting) ermöglicht eine Narrowing Reference Conversion:

```
MessagePost m = (MessagePost) p;
```

aber nur wenn der IPost auch wirklich ein MessagePost ist!

- Ein Typ (Klasse) in Klammern.
- Lösung für den „Verlust von Typinformationen“
- Das Objekt wird dabei nicht verändert
- Zur Laufzeit wird eine Überprüfung vorgenommen, ob das Objekt auch wirklich von dem erwarteten Typ ist:
  - `ClassCastException` wenn dies nicht der Fall ist!
  - Ohne weitere Vorkehrungen führen alle Exceptions zu einem irregulären Programmabbruch.
- Casting vermeiden!

# Übung 3

- Überprüfen Sie die Aussagen der Folie 36 und 37 in der Direkteingabe von BlueJ.
- Nutzen sie die `display` Methode um den Erfolg der Zuweisungen zu überprüfen.

# Zusammenfassung

- Vererbung erlaubt die Definition von Klassen als Erweiterung einer bereits existierenden Klasse
- Vererbung
  - Vermeidet Codeduplikation innerhalb der Klassenhierarchie
  - Ermöglicht Codewiederverwendung
  - Vereinfacht den Code
  - Vereinfacht Wartung und Erweiterung
- Variablen können auch Instanzen von Subtypen beinhalten
- Subtypen können im Allgemeinen immer dort verwendet werden wo Instanzen von Supertypen verwendet werden (Substitution)

# Abschließende Übung

- Setzen sie das Klassendiagramm auf Folie 33 um.
- Testen Sie das Ergebnis.



# Fragen?

Für weitere Fragen im  
Nachgang können Sie mich  
gerne über Moodle oder via  
E-Mail kontaktieren!

# Konzepte: Zusammenfassung I

- **Vererbung** Vererbung erlaubt uns, eine Klasse als eine Erweiterung einer anderen zu definieren.
- **Superklasse** Eine Superklasse ist eine Klasse, die von anderen Klassen erweitert wird.
- **Subklasse** Eine Subklasse ist eine Klasse, die eine andere Klasse erweitert bzw. von dieser Klasse erbt. Sie erbt alle Exemplarvariablen und Methoden von ihrer Superklasse.
- **Konstruktoren der Oberklasse** Im Konstruktor einer Subklasse muss immer als erste Anweisung der Konstruktor der Superklasse aufgerufen werden. Wenn im Quelltext kein solcher Aufruf angegeben ist, versucht Java automatisch, einen parameterlosen Aufruf einzufügen.
- **Subtypen** Analog zur Klassenhierarchie bilden die Objekttypen eine Typhierarchie. Der Typ, der durch eine Subklasse definiert ist, ist ein Subtyp des Typs, der durch die zugeordnete Superklasse definiert wird.
- **Variablen und Subtypen** Eine Variable kann ein Objekt halten, dessen Typ entweder gleich dem deklarierten Typ der Variablen oder ein beliebiger Subtyp des deklarierten Typs ist.

# Vererbung

**Vererbung erlaubt uns, eine Klasse als eine Erweiterung einer anderen zu definieren.**

# Superklasse

**Eine Superklasse ist eine Klasse, die von anderen Klassen erweitert wird.**

# Subklasse

**Eine Subklasse ist eine Klasse, die eine andere Klasse erweitert bzw. von dieser Klasse erbt. Sie erbt alle Exemplarvariablen und Methoden von ihrer Superklasse.**

# Konstruktor der Oberklasse

**Im Konstruktor einer Subklasse muss immer als erste Anweisung der Konstruktor der Superklasse aufgerufen werden. Wenn im Quelltext kein solcher Aufruf angegeben ist, versucht Java automatisch, einen parameterlosen Aufruf einzufügen.**

# Subtypen

**Analog zur Klassenhierarchie bilden die Objekttypen eine Typhierarchie. Der Typ, der durch eine Subklasse definiert ist, ist ein Subtyp des Typs, der durch die zugeordnete Superklasse definiert wird.**

# Variablen und Subtypen

**Eine Variable kann ein Objekt halten, dessen Typ entweder gleich dem deklarierten Typ der Variablen oder ein beliebiger Subtyp des deklarierten Typs ist.**



# Syntaxdiagramme I

Verallgemeinert (Klassendefinition, Vererbung)

