Überdeckte Methoden bei der Erweiterung von Klassen

- Ausgabe: Version 2
- Überdeckte Methoden
- Korrekter Gebrauch der Erweiterungs-/Vererbungsbeziehung

Frage

Warum ist die Variante, mit einer *druckeAufDieKonsole*-Methode Informationen zu Objekten auf der Konsole auszugeben, relativ „ungeschickt“?

Man benötigt für jedes „Ausgabeziel“ eine neue Methode

Klasse „Versicherter“

```
public void druckeAufDieKonsole() {
    println(vorname + " " + name);
    println("  Mitgliedsnummer: " + mitgliedsnummer);
}

public void druckeInEineDatei(BufferedWriter out) throws IOException {
    out.write(vorname + " " + name);
    out.newLine();
    out.write("  Mitgliedsnummer: " + mitgliedsnummer);
    out.newLine();

}

public void schreibeInEinTextfeld(JTextArea feld) {
    String text = vorname + " " + name + "\n"
            + "  Mitgliedsnummer: " + mitgliedsnummer;
    feld.setText(text);
}
```

Man benötigt für jedes „Ausgabeziel“ eine neue Methode

```
Versicherter mitglied = verwaltung.suche("Dampf", "Hans");
mitglied.druckeAufDieKonsole();

try (BufferedWriter out = new BufferedWriter(new FileWriter("proto.txt"))) {
    mitglied.druckeInEineDatei(out);
} catch (IOException e) {
}

JFrame fr = new JFrame();
JTextArea feld = new JTextArea(5, 20);
fr.add(feld);
fr.pack();
fr.setLocationRelativeTo(null);
mitglied.schreibeInEinTextfeld(feld);
fr.setVisible(true);
```

erter“

)Exception {

```
        out.newLine();
        out.write("  Mitgliedsnummer: " + mitgliedsnummer);
        out.newLine();

    }

    public void schreibeInEinTextfeld(JTextArea feld) {
        String text = vorname + " " + name + "\n"
                + "  Mitgliedsnummer: " + mitgliedsnummer;
        feld.setText(text);
    }
```

Man benötigt für jedes „Ausgabeziel“ eine neue Methode

```
Versicherter mitglied = verwaltung.suche("Dampf", "Hans");
mitglied.druckeAufDieKonsole();

try (BufferedWriter out = new BufferedWriter(new FileWriter("proto.txt"))) {
    mitglied.druckeInEineDatei(out);
} catch (IOException e) {
}
```

Bessere Lösung:
den Versicherten in einen String „wandeln“ (Methode *toString()*).
Dieser String kann überall verwendet werden!

```
}

public void schreibeInEinTextfeld(JTextArea feld) {
    String text = vorname + " " + name + "\n"
            + "  Mitgliedsnummer: " + mitgliedsnummer;
    feld.setText(text);
}
```

Eine Methode für jedes Ausgabeziel

Bessere Lösung:
den Versicherten in einen String „wandeln“
(Methode *toString()*).
Dieser String kann überall
verwendet werden!

```
public String toString() {
    return vorname + " " + name + "\n"
        + "  Mitgliedsnummer: " + mitgliedsnummer;
}
```

Eine Methode für jedes Ausgabeziel

```
Versicherter mitglied = verwaltung.suche("Dampf", "Hans");
println(mitglied.toString());

try (BufferedWriter out = new BufferedWriter(new FileWriter("proto.txt"))) {
    out.write(mitglied.toString());
    out.newLine();
} catch (IOException e) {
}

JFrame fr = new JFrame();
JTextArea feld = new JTextArea(5, 20);
fr.add(feld);
fr.pack();
fr.setLocationRelativeTo(null);
feld.setText(mitglied.toString());
fr.setVisible(true);
```

```
public String toString() {
    return vorname + " " + name + "\n"
        + "  Mitgliedsnummer: " + mitgliedsnummer;
}
```

Wir arbeiten also mit *toString()*-Methoden 1/3

```
public class Versicherter {

    private String name, vorname;
    private int mitgliedsnummer;

    public Versicherter( String name, String vorname, int mitgliedsnummer ) {
        this.name = name;
        this.vorname = vorname;
        this.mitgliedsnummer = mitgliedsnummer;
    }

    public boolean heißt( String gesuchterName, String gesuchterVorname ) {
        return name.equals(gesuchterName) && vorname.equals(gesuchterVorname);
    }

    public String toString() {
        return vorname + " " + name + "\n"
            + "  Mitgliedsnummer: " + mitgliedsnummer;
    }

}
```

Wir arbeiten also mit *toString()*-Methoden 2/3

```
public class Student extends Versicherter {

    private String hochschule;
    private String fach;
    private int matrikelnummer;

    public Student( String name, String vorname, int mitgliedsnummer,
            String hochschule, String fach, int matrikelnummer ) {
        super(name, vorname, mitgliedsnummer);
        this.hochschule = hochschule;
        this.fach = fach;
        this.matrikelnummer = matrikelnummer;
    }

    public String toString() {
        return super.toString() + "\n"
            + "  studiert " + fach + " an der " + hochschule + "\n"
            + "  Matrikelnummer: " + matrikelnummer;
    }

}
```

neues Attribut
(damit es interessanter wird)

neuer Parameter,
neue Zuweisung

Mitbenutzen der Ausgabe
des *Versicherten*

erweiterte Ausgabe

Wir arbeiten also mit *toString()*-Methoden 3/3: neue Klasse *InformatikStudent*

```
public class InformatikStudent extends Student {

    private String benutzerKennzeichen;

    public InformatikStudent( String name, String vorname, int mitgliedsnummer,
            String hochschule, int matrikelnummer ) {
        super(name, vorname, mitgliedsnummer, hochschule, "Informatik", matrikelnummer);
        this.benutzerKennzeichen = name.toLowerCase() + mitgliedsnummer;
    }

    public String toString() {
        return super.toString() + "\n"
            + "  Kennzeichen: " + benutzerKennzeichen;
    }

}
```

Für *InformatikStudenten* bleibt das Fach immer gleich

Der Account-Name wird eindeutig bestimmt

Mitbenutzen der Ausgabe des *Studenten*

erweiterte Ausgabe

```java
public class Student extends Versicherter {

    private String hochschule;
    private String fach;
    private int matrikelnummer;

    //...

    public String toString() {
        return super.toString() + "\n"
            + "  studiert " + fach + " an der " + hochschule + "\n"
            + "  Matrikelnummer: " + matrikelnummer;
```

```java
public class Versicherter {

    private String name, vorname;
    private int mitgliedsnummer;

    //...

    public String toString() {
        return vorname + " " + name + "\n"
            + "  Mitgliedsnummer: " + mitgl
    }

}
```

```java
public class InformatikStudent extends Student {

    private String benutzerKennzeichen;

    //...

    public String toString() {
        return super.toString() + "\n"
            + "  Kennzeichen: " + benutzerKennzeichen;
    }
```

```java
verwaltung.legeVersichertenAn(
        new InformatikStudent("Dampf", "Hans", 1, "HS MA", 1720001) );

Versicherter mitglied = verwaltung.suche("Dampf", "Hans");
println(mitglied.toString());
```

VM: dynamischer Typ *InformatikStudent*

(Teil-)String:
Vorname, Name
Mitgliedsnummer

```
public class Student extends Versicherter {

    private String hochschule;
    private String fach;
    private int matrikelnummer;

    //...

    public String toString() {
        return super.toString() + "\n"
            + "  studiert " + fach + " an der " + hochschule + "\n"
            + "  Matrikelnummer: " + matrikelnummer;
```

```
public class Versicherter {

    private String name, vorname;
    private int mitgliedsnummer;

    //...

    public String toString() {
        return vorname + " " + name + "\n"
            + "  Mitgliedsnummer: " + mitgl
    }

}
```

```
public class InformatikStudent extends Student {

    private String benutzerKennzeichen;

    //...

    public String toString() {
        return super.toString() + "\n"
            + "  Kennzeichen: " + benutzerKennzeichen;
    }
```

```
verwaltung.legeVersichertenAn(
        new InformatikStudent("Dampf", "Hans", 1, "HS MA", 1720001) );

Versicherter mitglied = verwaltung.suche("Dampf", "Hans");
println(mitglied.toString());
```

VM: dynamischer Typ *InformatikStudent*

(Teil-)String:
Vorname, Name
Mitgliedsnummer

```
public class Student extends Versicherter {

    private String hochschule;
    private String fach;
    private int matrikelnummer;

    //...

    public String toString() {
        return super.toString() + "\n"
            + "  studiert " + fach + " an der " + hochschule + "\n"
            + "  Matrikelnummer: " + matrikelnummer;
```

```
public class Versicherter {

    private String name, vorname;
    private int mitgliedsnummer;

    //...

    public String toString() {
        return vorname + " " + name + "\n"
            + "  Mitgliedsnummer: " + mitgl
    }

}
```

```
public class InformatikStudent extends Student {

    private String benutzerKennzeichen;

    //...

    public String toString() {
        return super.toString() + "\n"
            + "  Kennzeichen: " + benutzerKennzeichen;
    }
```

```
verwaltung.legeVersichertenAn(
        new InformatikStudent("Dampf", "Hans", 1, "HS MA", 1720001) );

Versicherter mitglied = verwaltung.suche("Dampf", "Hans");
println(mitglied.toString());
```

VM: dynamischer Typ *InformatikStudent*

(Teil-)String:
- Vorname, Name
- Mitgliedsnummer
- Fach, Hochschule
- Matrikelnummer

```java
public class Student extends Versicherter {

    private String hochschule;
    private String fach;
    private int matrikelnummer;

    //...

    public String toString() {
        return super.toString() + "\n"
            + "  studiert " + fach + " an der " + hochschule + "\n"
            + "  Matrikelnummer: " + matrikelnummer;
```

```java
public class Versicherter {

    private String name, vorname;
    private int mitgliedsnummer;

    //...

    public String toString() {
        return vorname + " " + name + "\n"
            + "  Mitgliedsnummer: " + mitgl
    }

}
```

```java
public class InformatikStudent extends Student {

    private String benutzerKennzeichen;

    //...

    public String toString() {
        return super.toString() + "\n"
            + "  Kennzeichen: " + benutzerKennzeichen;
    }
```

```java
verwaltung.legeVersichertenAn(
        new InformatikStudent("Dampf", "Hans", 1, "HS MA", 1720001) );

Versicherter mitglied = verwaltung.suche("Dampf", "Hans");
println(mitglied.toString());
```

VM: dynamischer Typ *InformatikStudent*

(Teil-)String:
- Vorname, Name
- Mitgliedsnummer
- Fach, Hochschule
- Matrikelnummer
- Benutzer-Kennzeichen

```
public class Student extends Versicherter {

    private String hochschule;
    private String fach;
    private int matrikelnummer;

    //...

    public String toString() {
        return super.toString() + "\n"
            + "  studiert " + fach + " an der " + hochschule + "\n"
            + "  Matrikelnummer: " + matrikelnummer;
```

```
public class Versicherter {

    private String name, vorname;
    private int mitgliedsnummer;

    //...

    public String toString() {
        return vorname + " " + name + "\n"
            + "  Mitgliedsnummer: " + mitgl
    }

}
```

```
public class InformatikStudent extends Student {

    private String benutzerKennzeichen;

    //...

    public String toString() {
        return super.toString() + "\n"
            + "  Kennzeichen: " + benutzerKennzeichen;
    }
```

```
verwaltung.legeVersichertenAn(
        new InformatikStudent("Dampf", "Hans", 1, "HS MA", 1720001) );

Versicherter mitglied = verwaltung.suche("Dampf", "Hans");
println(mitglied.toString());
```

VM: dynamischer Typ *InformatikStudent*

Ausgabe des Strings (jetzt erst!):

- Vorname, Name
- Mitgliedsnummer
- Fach, Hochschule
- Matrikelnummer
- Benutzer-Kennzeichen

```
public class Student extends Versicherter {

    private String hochschule;
    private String fach;
    private int matrikelnummer;

    //...

    public String toString() {
        return super.toString() + "\n"
            + "  studiert " + fach + " an der " + hochschule + "\n"
            + "  Matrikelnummer: " + matrikelnummer;
```

```
public class Versicherter {

    private String name, vorname;
    private int mitgliedsnummer;

    //...

    public String toString() {
        return vorname + " " + name + "\n"
            + "  Mitgliedsnummer: " + mitgl
    }

}
```

```
public class InformatikStudent extends Student {

    private String benutzerKennzeichen;

    //...

    public String toString() {
        return super.toString() + "\n"
            + "  Kennzeichen: " + benutzerKennzeichen;
    }
```

```
verwaltung.legeVersichertenAn(
        new InformatikStudent("Dampf", "Hans", 1, "HS MA", 1720001) );

Versicherter mitglied = verwaltung.suche("Dampf", "Hans");
println(mitglied.toString());
```

Überdeckte Methoden

- Gibt es in einer Klasse (z.B. *Versicherter*) eine Methode (z.B. *toString*) und wird in einer anderen Klasse (z.B. *Student*), welche die erste (*Versicherter*) erweitert, ebenfalls eine Methode mit
  - gleichem *Namen (toString*) und
  - gleicher (hier: *leerer) Parameterliste*

  vereinbart, dann sagt man, die Methode (*toString*) der Unterklasse (*Student) **überdeckt*** die Methode (*toString*) der Oberklasse (*Versicherter*).

- Am Beispiel:

  new Student().toString()

  ruft die Methode *toString* der Klasse *Student* auf.
  Gäbe es keine überdeckende Methode *toString* in *Student*, würde automatisch die Methode *toString* der Klasse *Versicherter* ausgeführt werden: Das *Student*-Objekt *ist ja ein* spezielles Versicherter-Objekt, also ist der Aufruf zulässig (der Compiler überprüft die Zulässigkeit).

Allgemein: überdeckte Methoden

- Gibt es in einer Klasse *O* eine Methode *m* und wird in einer Unterklasse *U*, die *O* erweitert, ebenfalls eine Methode mit
  - gleichem *Namen* und
  - gleicher *Parameterliste*

  (Anmerkung: Unterschiedliche Rückgabetypen sind nicht zulässig!)

  vereinbart, dann sagt man, die Methode *m* der Klasse *U* ***überdeckt*** die Methode *m* der Klasse *O*.

- Verallgemeinert:

  new U().m()

  ruft die Methode *m* der Klasse *U* auf.
  Gäbe es keine überdeckende Methode *m* in *U*, würde automatisch die Methode *m* der Klasse *O* ausgeführt werden: das *U*-Objekt *ist ja ein* erweitertes/spezielles O-Objekt, also ist der Aufruf zulässig.

Zur Erinnerung

Feststellung

Wenn eine Klasse eine andere Klasse *erweitert*, dann bedeutet das, die erweiterte Klasse *kann alles, was die ursprüngliche kann* plus einiges mehr!

Ein Objekt der erweiterten Klasse *ist gleichzeitig* ein spezialisiertes Objekt der ursprünglichen Klasse!

In Java:
Man kann die ursprünglichen Methoden auch auf die neuen Objekte anwenden!

Überdeckte Methoden

- Nochmal:
  Gibt es in einer Klasse *O* eine Methode *m* und wird in einer Unterklasse *U*, die *O* erweitert, ebenfalls eine Methode mit
  - gleichem Namen und
  - gleicher Parameterliste

  vereinbart, dann sagt man, die Methode *m* der Klasse *U* **überdeckt** die Methode *m* der Klasse *O*.

- Soll die Methode *m* in *U* die Methode *m* in *O* nicht komplett überdecken, sondern nur *ergänzen*, kann diese Methode mittels

  super.m(...)

  aufgerufen werden.

Erläuterung: Überdeckte *Methoden*

- Beim Aufruf einer (nicht-statischen) Methode in Java wird *zunächst* in der *Klasse des Objekts*, von dem die Methode aufgerufen wird, *dann* in ihrer *direkten Oberklasse*, *dann* in *deren direkter Oberklasse* etc. eine Methode gesucht, die zur angegebenen Parameterliste passt
- Der Aufrufende sieht nicht, in welcher der Klassen die Methode tatsächlich definiert ist
  (und braucht es nicht zu sehen! und will es gar nicht wissen!)
- Der *Compiler* stellt sicher, dass es (zumindest) eine passende Methode gibt (sonst bricht die Übersetzung ab)
- Dieser Aufrufmechanismus ermöglicht es, gemeinsame Code-Anteile von Klassen in (eine) gemeinsame Oberklasse(n) auszugliedern

Überdeckung von *Feldern*

- Es ist (technisch) möglich, dass eine (Unter-) Klasse ein Feld mit einem Namen besitzt, der bereits für ein Feld in einer Oberklasse dieser Klasse vergeben ist
- Dieses Feld ***überdeckt*** (wie bei Methoden) das Feld der Oberklasse innerhalb der Klasse, welche die neue Deklaration vornimmt und in allen ihren Unterklassen

  In der Oberklasse selbst ist natürlich nur deren Feld sichtbar
- Das Überdecken ist unabhängig vom Typ der beteiligten Felder

Achtung:

Ein solcher Konflikt *darf* eigentlich *niemals* vorkommen:

Wenn eine Klasse als Unterklasse einer anderen definiert ist, so ist sie (*per definitionem*) eine Spezialisierung dieser Klasse

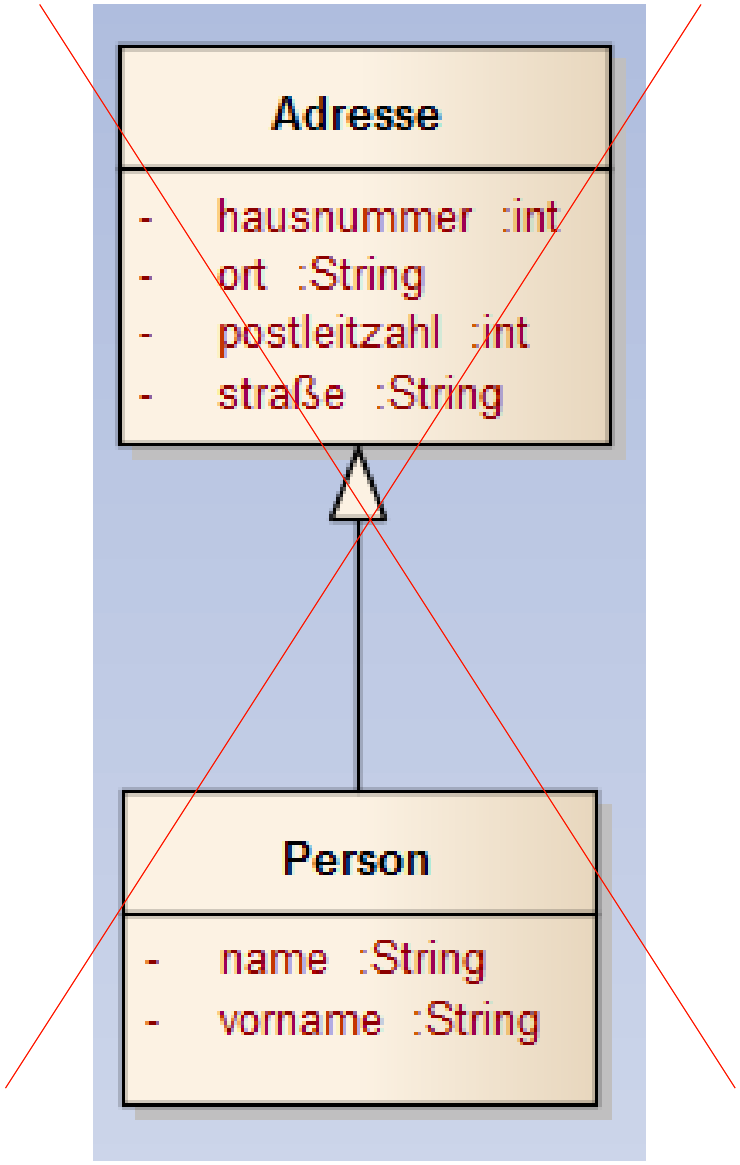
Damit sollte, sofern „vernünftige“ Namen verwendet werden, das Feld der Unterklasse die gleiche Bedeutung haben wie das Feld der Oberklasse, also ist die erneute Definition unnötig (falsch?)! Beispiel: Warum sollte der *InformatikStudent* eine eigene (weitere?) *Matrikelnummer* benötigen?

Korrekter Gebrauch der Vererbungsbeziehung als Erweiterung / Spezialisierung

- Die Erweiterung von Programmen um zusätzliche Klassen ist vergleichsweise einfach, wenn die Vererbungsbeziehung als reine Erweiterung/Spezialisierung eingesetzt wird
- Wird die Vererbungsbeziehung zu anderen Zwecken *missbraucht*, zum Beispiel, *um Schreibarbeit* bei Klassen *zu sparen*, zwischen denen keine Spezialisierungsbeziehung besteht, kommt es sehr oft zu Problemen, wenn das ursprüngliche Programm erweitert werden soll
- Wir erinnern uns:
  50-75% aller durchgeführten Programmierarbeiten sind sogenannte „Wartung und Pflege“, d.h. Verbessern, Umbauen, Erweitern oder Weiterentwickeln von existierendem Programmcode

→ Es lohnt sich also, von Beginn an über eine gute Programmstruktur nachzudenken, welche die spätere Wartung erleichtert!

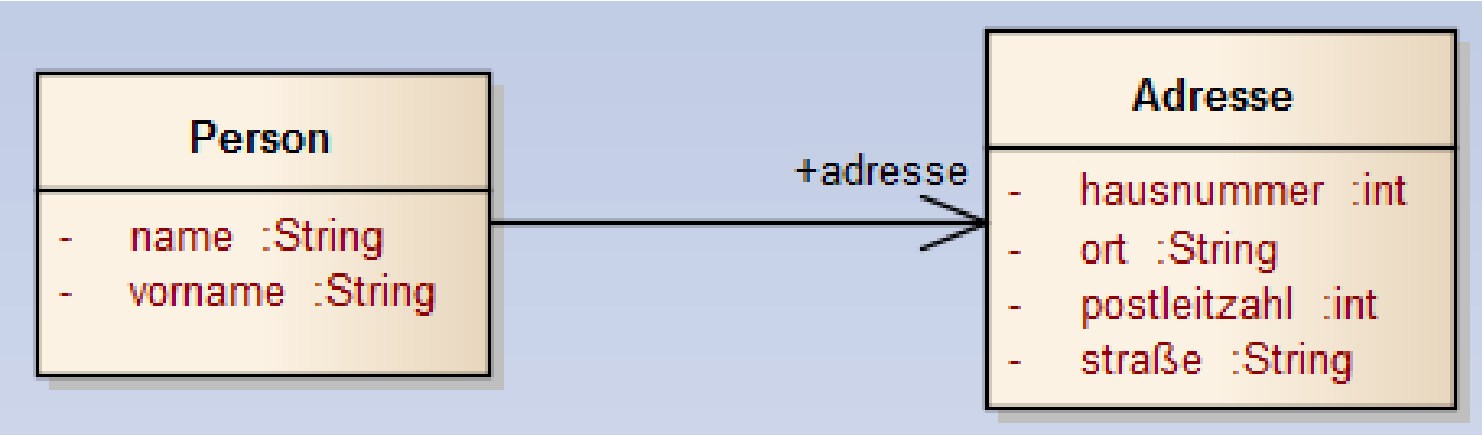
Beispiel für einen *nicht vernünftigen* Gebrauch



Wird die Vererbungsbeziehung zu anderen Zwecken *missbraucht*, zum Beispiel, *um Schreibarbeit* bei Klassen *zu sparen*, zwischen denen keine Spezialisierungsbeziehung besteht, kommt es sehr oft zu Problemen, wenn das ursprüngliche Programm erweitert werden soll

Korrekte Form

Die Person *hat* eine Adresse (*nicht*: sie ist eine Adresse),
es handelt sich um eine Kompositionsbeziehung



Klassenhierarchien

- Wie wir gesehen haben, ist die Vererbungsbeziehung *transitiv*, d.h. sie bezieht sich nicht nur auf eine direkte Oberklasse, sondern geht auch über mehrere (beliebig viele) Hierarchieebenen hinweg

  Beispiel:
  Die Klasse *InformatikStudent* besitzt nicht nur alle Felder ihrer (direkten) Oberklasse *Student*, sondern auch alle Felder ihrer (nicht direkten) Oberklasse *Versicherter*

- Bemerkung:
  In der Praxis zeigt sich, dass Klassenhierarchien mit mehr als 5-6 Ebenen schwer verständlich werden