Vererbung

- Unterklassen einer Klasse
- Vererbung von Methoden und Instanzvariablen
- Überschreiben von Methoden
- Vererbung als Realisierung einer “is-a” Beziehung.

Beispiel: Bankkonten

```
public class Sparkonto extends Bankkonto {
  /* Neue Instanzvariablen, z.B. Zinssatz */
  /* Neue Methoden, z.B. Zinsen berechnen */
  /* Neue Konstruktoren, z.B. mit Angabe des Zinssatzes */
}
```

Die bisherigen Methoden bleiben verfügbar, als hätte man sie in die Definition von `Sparkonto()` hineinkopiert.

Bisherige private Instanzvariablen sind nicht sichtbar.

Für Konstruktoren gilt eine besondere Regel.

Sparkonten

```
public class Sparkonto extends Bankkonto {
    private double zinssatz;

    public void zinsenAnrechnen() {
        double zinsen = this.getKontostand() * zinssatz / 10
        this.einzahlen(zinsen);
    }

    public Sparkonto(double satz) {
         zinssatz = satz;
    }
}
```

Man könnte “`this.`” hier auch weglassen.

Verwendung

```
Sparkonto johannasSpar = new Sparkonto(1.25);
Bankkonto matthiasGiro = new Bankkonto(10);

johannasSpar.einzahlen(200.0);
johannasSpar.zinsenAnrechnen();
System.out.println("Johannas Sparkonto: " +
                   johannasSpar.getKontostand());
```

Ausgabe:

```
Johannas Sparkonto: 207.5625
```

Erinnerung: Der Konstruktor `Bankkonto()` setzt den Kontostand auf 5,– (Geschenk der Sparkasse).

Funktionsweise

Beim Aufruf

```
johannasSpar.zinsenAnrechnen();
```

wird folgender Code ausgeführt:

```
double zinsen = johannasSpar.getKontostand() *
                                    zinssatz / 100.0;
johannasSpar.einzahlen(zinsen);
```

Terminologie

`Bankkonto` ist die Oberklasse (*superclass*) von `Sparkonto`.

`Sparkonto` erbt von (*inherits from*) `Bankkonto`.

`Sparkonto` ist eine Unterklasse (*subclass*) von `Bankkonto`.

In Java hat jede Klasse nur eine Oberklasse.

Eine Klasse kann aber mehrere Unterklassen haben.

Z.B. bei Bankkonto: Girokonto, Tagesgeldkonto, etc.

Auch von Unterklassen kann man weiter ererben: Prämiensparkonto als Unterklasse von Sparkonto, etc.

Subtyping

Ein Objekt der Unterklasse kann immer dort verwendet werden, wo ein Objekt der Oberklasse erwartet wird.

Beispiel:

```
Bankkonto matthiasGiro = new Bankkonto();
matthiasGiro.ueberweisen(200, johannasSpar);
```

Erinnerung:

```
public void ueberweisen(double betrag, Bankkonto empfaenger)
```

Zwischen der Unterklasse und der Oberklasse besteht eine "ist-ein"-Beziehung ("*is a*" *relationship*).

In den Anwendungen ergibt sich oft eine komplexe Klassenhierarchie.

Klassenhierarchien

- – Thecodonten, Pterosaurier, Dinosaurier, Crocodilier sind “Unterklassen” der Archosaurier.
  – Saurischier und Ornithischier sind Unterklassen der Dinosaurier.
- – JPanel, JTextcomponent, JLabel, AbstractButton sind Unterklassen von JComponent
  – JTextField, JTextArea sind Unterklassen von JTextComponent
  – JToggleButton und JButton sind Unterklassen von AbstractButton
  – JCheckBox und JRadioButton sind Unterklassen von JToggleButton

UML-Notation: Pfeil mit hohler Dreieckspitze ($\multimap\!\triangleright$) von Unterklasse zu Oberklasse.

Vererbung von Instanzvariablen

Alle Instanzvariablen der Unterklasse werden vererbt,

Private Instanzvariablen sind aber in der Unterklasse nicht sichtbar.

Auf sie kann allerdings mit ererbten Methoden der Oberklasse zugegriffen werden.

Ein Sparkonto hat also die Instanzvariablen `kontostand` und `zinssatz`. Die neugeschriebenen Methoden können aber nur `zinssatz` beeinflussen.

Man kann eine Instanzvariable als `protected` kennzeichnen. Dann ist sie auch in den Unterklassen sichtbar. Horstmann rät aus diversen Gründen davon ab.

Konstruktoren der Unterklasse

Konstruktoren der Oberklasse werden mit `super(...)` bezeichnet.

Man sollte einen Konstruktor der Oberklasse zu Beginn des Konstruktors der Unterklasse aufrufen um die ererbten privaten Instanzvariablen geeignet zu initialisieren.

Tut man das nicht (wie wir im Beispiel), so wird der Defaultkonstruktor (ohne Parameter) der Oberklasse automatisch eingefügt.

Wir könnten z.B. schreiben:

```
public Sparkonto(double betrag, double satz) {
   super(betrag);
   zinssatz = satz;
}
```

Überschreiben

Definiert man in der Unterklasse eine Methode, die es in der Oberklasse schon gibt (Methodenname und Parameterzahl und -typen stimmen überein), so wird die ererbte Methode überschrieben (engl. *overriding* = “sich hinwegsetzen über”).

Die neudefinierte Methode ersetzt dann die alte überall, d.h.

auch in ererbten Methoden, die die überschriebene Methode verwenden

Beispiel Girokonto

```
public class Girokonto extends Bankkonto {
    private int anzahlTransaktionen;
    private double gebuehrensatz;

    public Girokonto(double satz) {
        super(0.0);
        anzahlTransaktionen = 0;
        gebuehrensatz = satz;
    }

    public void einzahlen(double betrag) {
        super.einzahlen(betrag);
        anzahlTransaktionen++;
    }

    public void abheben(double betrag) {
```

Beispiel Girokonto

```
        super.abheben(betrag);
        anzahlTransaktionen++;
    }

    public void gebuehrenAbziehen() {
        double gebuehren = anzahlTransaktionen * gebuehrensa
        super.abheben(gebuehren);
        anzahlTransaktionen = 0;
    }
}
```

Bemerkungen

- Die Pseudovariable `super` bezeichnet das aktuelle Objekt aufgefasst als Objekt der Oberklasse.
- Hätten wir geschrieben

  ```
  this.abheben(gebuehren);
  ```

  so wären für das Abziehen der Gebühren gleich wieder welche fällig geworden. (Entspricht wahrscheinlich sogar der Realität!).
- Wir können in `Girokonto` nicht schreiben

  ```
  kontostand = kontostand - gebuehren;
  ```

  da `kontostand` nicht sichtbar ist.
- Wir könnten natürlich eine neue Instanzvariable mit Namen `kontostand` deklarieren, dann hätte jedes Objekt aber zwei verschiedende Instanzvariablen des Namens `kontostand`.

Geschachtelte Vererbung

Ist A Unterklasse von B, so kann man ohne weiteres eine Unterklasse C von B definieren.

Jedes Objekt der Klasse C ist dann automatisch ein Objekt der Klasse B und als solches auch ein Objekt der Klasse A.

Die Oberklasse von C ist aber nur B.

Festgeldkonto

```
public class Festgeldkonto extends Sparkonto {
    int restJahre;
    double strafgebuehr;

    public void zinsenBerechnen() {
        restJahre = restJahre--;
        super.zinsenAnrechnen();
    }

    public void abheben(double betrag) {
        if (restJahre > 0)
            super.abheben(strafgebuehr);
        super.abheben(betrag);
    }

    Festgeldkonto(int jahre, int gebuehr, double zinssatz) {
```

Festgeldkonto

```
        super(zinssatz);
        strafgebuehr = gebuehr;
        restJahre = jahre;
    }
}
```

Polymorphie

Jedes Objekt gehört zu einer festen Klasse.

Welche Methode bei einem Objekt aufgerufen wird, richtet sich immer nach seiner Klasse, auch dann, wenn es per Subtyping als Objekt der Oberklasse benutzt wird.

Das bezeichnet man als Polymorphie. Aber Vorsicht: dieser Begriff wird auch in anderen Zusammenhängen benutzt.

Beispiel: in

```
Girokonto matthiasGiro = new Girokonto(0.25);
Sparkonto johannasSpar = new Sparkonto(200.0);
johannasSpar.ueberweisen(100.0, matthiasGiro);
```

fallen bei `matthiasGiro` Gebühren an, obwohl die Methode `abheben` in `Bankkonto` keine Gebühren beaufschlagt.

Typecast und `instanceof`

Hat ein Ausdruck den Typ einer Oberklasse, so kann man ihn explizit auf die Unterklasse konvertieren:

```
Girokonto matthiasGiro = new Girokonto(0.25);
Bankkonto x = matthiasGiro;
Girokonto y = (Girokonto)x;
```

Wird so eine Typkonversion (*type cast* der Form `(A)e` ausgewertet, so wird überprüft, ob die tatsächliche Klasse von `e` Unterklasse (evtl. über mehrere Ecken) von `A` ist.

Falls ja, so wird in der Auswertung fortgefahren, falls nein, bricht das Programm ab.

Der Ausdruck `e instanceof A` hat den Wert `true` genau dann, wenn die tatsächliche Klasse von `e` unterhalb von `A` liegt.

Die Klasse `Object`

Die Klasse `Object` steht an der Spitze jeder Klassenhierarchie.

Sie definiert die Methoden

```
String toString() /* in einen String konvertieren */
boolean equals(Object other) /* auf Gleichheit testen */
Object clone() /* Kopie erzeugen */
```

Alle diese Methoden kann man in einer Klasse überschreiben.

Übungen

Was ist `b.getKontostand()` am Ende ?

```
Sparkonto b = new Sparkonto(10);
b.einzahlen(4995);
b.abheben(b.getKontostand() / 2);
b.zinsenAnrechnen();
```

Was sollte hier Unterklasse sein, was Oberklasse:

- Angestellter-Vorgesetzter
- Polygon- Dreieck
- Doktorstudent-Student
- Person-Student
- Angestellter-Doktorstudent
- Bankkonto-Girokonto

Übungen

- Fahrzeug-Pkw
- Fahrzeug-Van
- Pkw-Van
- Lkw-Fahrzeug

Die Klasse `Sub` sei Unterklasse von Sandwich. Welche der folgenden Zuweisungen sind erlaubt?

```
Sandwich x = new Sandwich();
Sub y = new Sub();
x = y;y = x;
y = new Sandwich();
x = new Sub();
```

Man überschreibe die Methode `toString` bei `Girokonto`.

Man implementiere eine Unterklasse `Quadrat` von `Rectangle`.

Behavioural Subtyping

Hat man in einer Oberklasse Methoden durch Vor- und Nachbedingungen spezifiziert, so sollten überschreibenden Versionen davon auch dieser Spezifikation genügen.

Der Vorteil ist dann, dass Spezifikationen für sämtliche ererbte Methoden fortgelten.

Ebenso sollten Invarianten, die in der Oberklasse vereinbart wurden auch von den Methoden der Unterklasse eingehalten werden.

Liegen all diese Bedingungen vor, so sagt man, die Unterklasse sei ein "*behavioural subtype* der Oberklasse.

Abstrakte Klassen und Methoden

Man kann eine Methodendeklaration in einer Klasse mit dem Schlüsselwort `abstract` kennzeichnen und keine Implementierung angeben.

Die gesamte Klasse muss dann auch das Schlüsselwort `abstract` tragen.

Man kann dann von dieser Klasse keine Instanzen erzeugen (`new` ist also verboten).

Allerdings kann man von der Klasse erben und dann die “abstrakten” Methoden konkret implementieren.

Beispiel

```
public abstract class AbstraktesBankkonto {

    public abstract void writeLog(String s);

    public void abheben(double betrag) {
        writeLog("Abhebung: "+betrag);
    }

    public void einzahlen(double betrag) {
        writeLog("Einzahlung: "+betrag);
    }

    public void ueberweisen(AbstraktesBankkonto b, double betrag){
        this.abheben(betrag);b.einzahlen(betrag);
    }
}
```

Beispiel

```
public class Bankkonto extends AbstraktesBankkonto {
    private static int naechsteNummer = 0;
    private int kontonummer;
    private double kontostand;

    public double getKontostand() {
        return kontostand;
    }
    public int getKontonummer() {
        return kontonummer;
    }

    public Bankkonto() {
        kontostand = 0;
        kontonummer = naechsteNummer;
        naechsteNummer++;
    }
```

Beispiel

```
    public void writeLog(String s) {
        System.out.println
            ("Kontonr.: " + getKontonummer() + "\n" +
             s +
             "\nNeuer Kontostand: " + getKontostand());
    }
    public void einzahlen(double betrag) {
        kontostand += betrag;
        super.einzahlen(betrag);
    }
    public void abheben(double betrag) {
        kontostand -= betrag;
        super.abheben(betrag);
    }
}
```

Verwendung

```
b.einzahlen(3);
c.einzahlen(6);
b.ueberweisen(c,4);
```

Ausgabe:

```
Kontonr.: 0
Einzahlung: 3.0
Neuer Kontostand: 3.0
Kontonr.: 1
Einzahlung: 6.0
Neuer Kontostand: 6.0
Kontonr.: 0
Abhebung: 4.0
Neuer Kontostand: -1.0
Kontonr.: 1
Einzahlung: 4.0
Neuer Kontostand: 10.0
```

Vererbung: Zusammenfassung

- Man kann zu einer gegebenen Klasse Unterklassen definieren:

  `class` *NameDerUnterklasse* `extends` *NameDerOberKlasse*{

  neue Konstruktoren

  neue Instanzvariablen

  neue Methoden}

- Konstruktoren, Instanzvar. Methoden der Oberklasse werden ererbt.
- Private Instanzvar. der Oberklasse sind in der Unterklasse nicht sichtbar.
- Man kann Methoden der Oberklasse überschreiben. Die neue Definition ersetzt die alte überall, auch in Definitionen von Methoden der Oberklasse.
- Mit `super` kann man auf Methoden und Konstruktoren der Oberklasse zugreifen.

Vererbung: Zusammenfassung

- Vererbung wird benutzt um “*is a*”-Beziehungen softwaretechnisch zu realisieren.
- Die UML-Notation für Vererbung ist ein `----|>` Pfeil von der Unter- zur Oberklasse.
- `Object` ist automatisch Oberklasse jeder Klasse. Die Methoden `clone, equals, toString` der Klasse `Object` können überschrieben werden.
- Behavioural subtyping: Spezifikationen aus der Oberklasse werden von der Unterklasse respektiert.
- Abstrakte Methoden in abstrakten Klassen werden nicht implementiert, sie müssen in konkreten Unterklassen überschrieben werden.