Objektorientierte Programmierung

Vererbung

Prof. Dr. Ulrike Hammerschall
Fakultät für Informatik und Mathematik

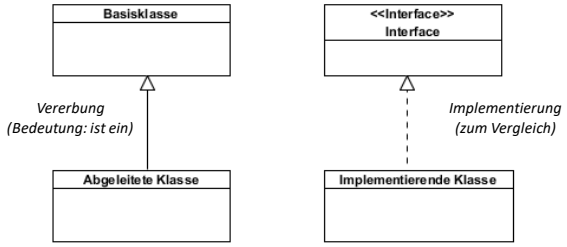
Themen

- Vererbung
- Abstrakte Basisklassen und Redefinition
- Die Klasse Object

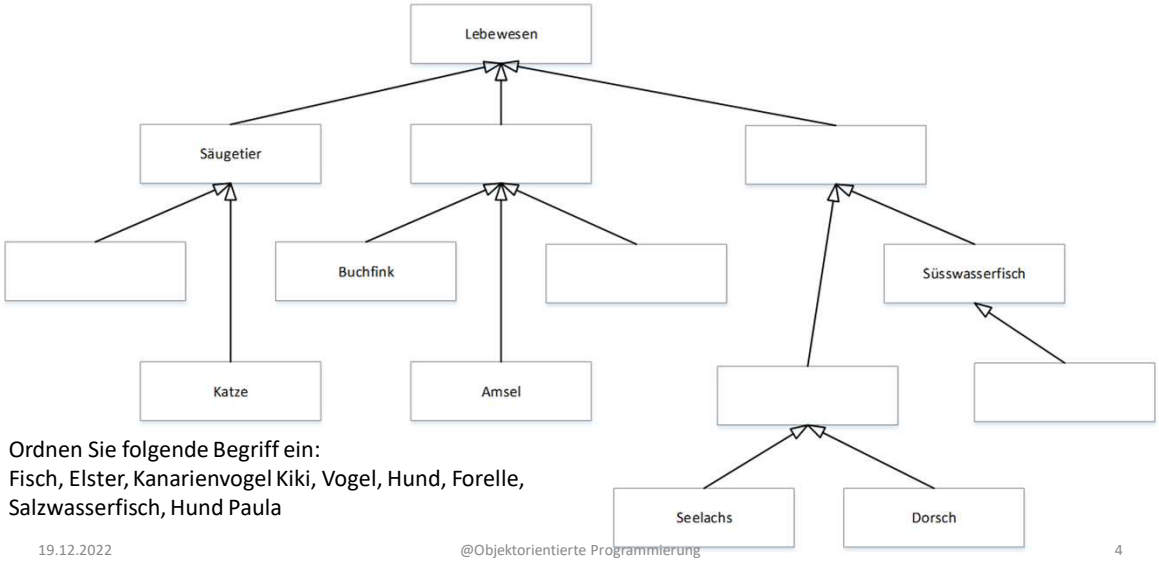
Vererbung in der Programmierung

- Beziehung zwischen Klassen (nicht zwischen Klassen und Objekten).
- Vererbt werden Eigenschaften (Attribute) und Verhalten (Methoden).

Darstellung in der UML

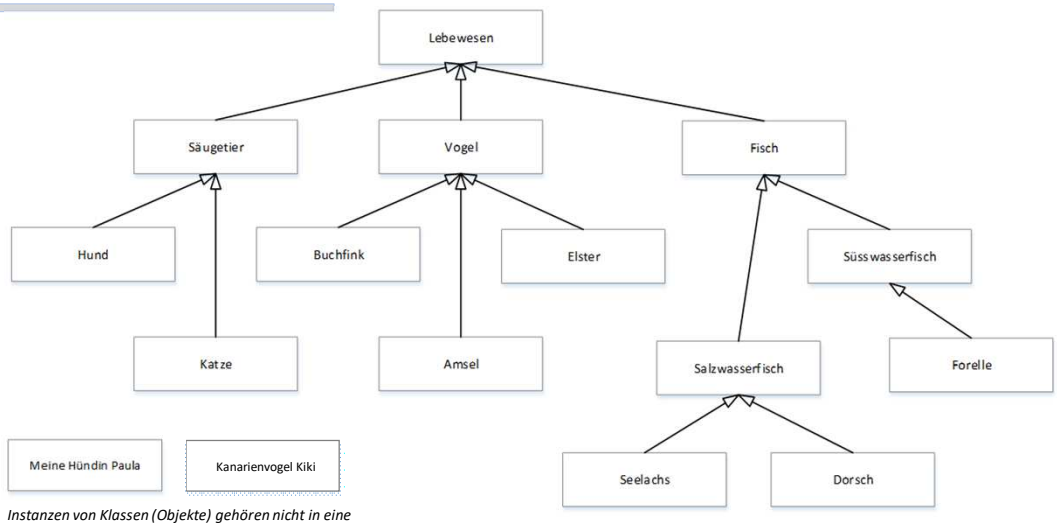


Vererbung als „ist ein“ - Beziehung



Ordnen Sie folgende Begriff ein:
Fisch, Elster, Kanarienvogel Kiki, Vogel, Hund, Forelle, Salzwasserfisch, Hund Paula

Vererbung: „ist ein“ Beziehung



Instanzen von Klassen (Objekte) gehören nicht in eine Vererbungshierarchie

Begrifflichkeiten

- Vererbung (Inheritance):
  - Name für das übergeordnete Konzept. Umfasst eine Menge von Eigenschaften, die mit Vererbung im Zusammenhang stehen.
  - Gilt generell für objektorientierte Programmiersprachen (z.B. Java, C++, C#) oder objektorientierte Methoden/Notationen (z.B. UML).
  - Synonyme: Ableitung, Spezialisierung (Generalisierung)
- Basisklasse (base class, super class):
  - Klasse von der abgeleitet wird.
- Abgeleitete Klasse (derived class):
  - Abgeleitete Klasse, die damit Eigenschaften der Basisklasse übernimmt.
- Instanz:
  - Entspricht einem Objekt einer Klasse. Zwischen einer Klasse und ihrer Instanz besteht KEINE Vererbungsbeziehung. Instanzen übernehmen jedoch die Eigenschaften ihrer Klassen auch bezogen auf Vererbung.

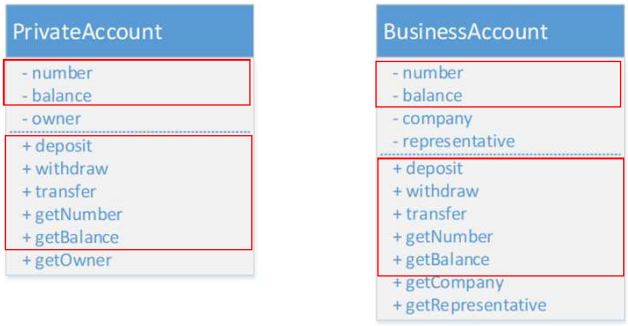
6

Vererbung in der Programmierung

- Ziel: Vermeidung von Redundanz:
  - Gemeinsame Attribute (Eigenschaften) und Methoden (Verhalten) verschiedener Klassen werden in eine Basisklasse verschoben.
  - Die abgeleiteten Klassen erben Attribute und Methoden von der Basisklasse.
  - Die abgeleiteten Klassen erweitern bzw. modifizieren Attribute und Methoden der Basisklasse.
  - Basisklasse und abgeleitete Klasse sind immer Klassen, keine Interfaces (Interfaces können jedoch selbst in einer Vererbungsbeziehung stehen).
- In Java wird Vererbung über das Schlüsselwort *extends* markiert.

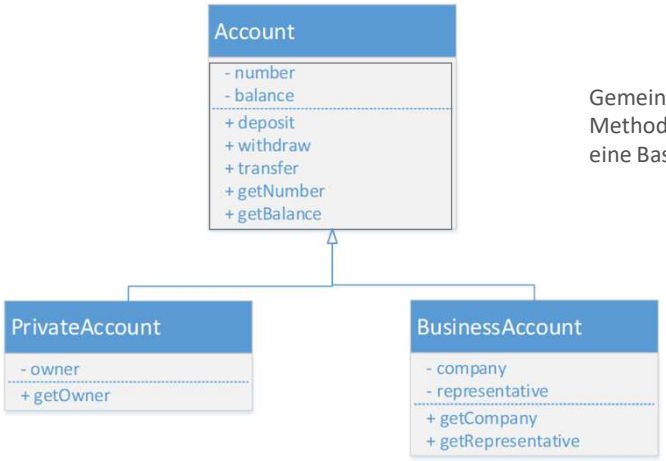
```
public class DerivedClass extends BaseClass {
        ...
}
```


Beispiel: Klassen für Konten ohne Vererbung



Redundante Attribute und Methoden

Beispiel: Klassen für Konten mit Vererbung



Gemeinsame Attribute und Methoden werden ausgelagert in eine Basisklasse Account

Differenzierende Attribute und Methoden bleiben in den abgeleiteten Klassen.

Was bedeutet Vererbung?

- Die abgeleitete Klasse erbt **alle**
  - nicht-privaten Objektvariablen und Objektmethoden der Basisklasse
- Die abgeleitete Klasse erbt **keine**
  - privaten Objektvariablen und Objektmethoden
  - Konstruktoren
- Für die Anwender einer Klasse sind geerbte Elemente und Elemente der Klasse selbst nicht unterscheidbar.

Basisklassen-Konstruktor

HM

- Jeder Konstruktor einer abgeleiteten Klasse ruft zuerst einen Konstruktor der Basisklasse auf.
- Dies erfolgt explizit oder implizit:
  - **explizit**: über den Aufruf super(..) im Konstruktor der abgeleiteten Klasse.
  - **implizit**: Der Default-Konstruktor der Basisklasse wird automatisch aufgerufen, wenn kein anderer Konstruktor ausgewählt wurde.
- Falls kein Default-Konstruktor existiert, muss explizit ein Custom-Konstruktor in der Basisklasse aufgerufen werden!

Basisklassen-Konstruktor

HM

- Bei der Verwendung von super() gilt:
  - kann nur im Konstruktor einer abgeleiteten Klasse verwendet werden.
  - nur ein Aufruf pro Konstruktor ist erlaubt (ähnlich zu this()).
  - super muss die erste Anweisung im Konstruktor-Rumpf sein (ähnlich zu this()).
- Folge: Das Basisklassenobjekt wurde bereits vollständig initialisiert, wenn der Konstruktor der abgeleiteten Klasse weiter ausgeführt wird.

Statischer und dynamischer Typ

HM

- Objekte abgeleiteter Klassen können
  - Variablen vom Typ der Basisklasse oder
  - Variablen vom Typ der abgeleiteten Klasse

  zugewiesen werden.
- Wie bei Interfaces Unterscheidung zwischen statischem Typ und dynamischem Typ. Objekte vom statischen Typ sind hier aber auch möglich.

```
Account account = new PrivateAccount();
```

➢ Kennt keine Methoden aus BusinessAccount, aber alle aus Account

```
Account account = new BusinessAccount();
```

➢ Kennt keine Methoden aus PrivateAccount, aber alle aus Account

```
Account account = new Account();
```

➢ Kennt keine Methoden aus BusinessAccount und PrivateAccount. Aber alle aus Account.

Sichtbarkeit

HM

- **private:** nur sichtbar für Methoden und Elemente innerhalb der Klasse
- **package:** sichtbar für Klassen, die im gleichen Package liegen.
- **public:** innerhalb der gesamten Anwendung für alle Klassen in allen Packages sichtbar.
- **protected:**
  - ➢ sichtbar für alle Klassen im gleichen Package, ob abgeleitet oder nicht.
  - ➢ sichtbar für abgeleitete Klassen auch in anderen Packages.

Themen

- Vererbung
- Abstrakte Basisklassen und Redefinition
- Die Klasse Object

Abstrakte Basisklassen (abstract base classes - ABC)

- Spezielle Art von Klassen (Eigenschaften von Klasse und Interface).
  - ➢ Von der Definition her ähnlich zu normalen Klassen.
  - ➢ Von der Verwendung her ähnlich zu Interfaces.
- Abstrakte Klassen sind Klassen mit ein paar Besonderheiten:
  - Was bleibt gleich?
    - Objektvariablen, Klassenvariablen, Konstanten, …
    - Konstruktoren
    - vollständig definierte Methoden,
  - Was sind die Besonderheiten?
    - Modifier *abstract* zur Kennzeichnung
    - Abstrakte Methoden (entsprechend zu Interfaces),

```
public abstract class Account {
    …
    public abstract String getOwnerName();
}
```

Besonderheiten abstrakter Klassen

- Von abstrakten Klassen abgeleitete konkrete Klassen:
  - erben alle **konkreten** Methoden (und nicht-private OV) der abstrakten Klassen.
  - müssen Implementierungen für alle **abstrakten** Methoden liefern.
- Abstrakte Klassen sind Basisklassen, die selbst nicht instanziiert werden können (trotz Konstruktoren).
- Zu jeder abstrakten Klasse muss mindestens eine konkrete Klasse oder eine Hierarchie konkreter Klassen existieren, die alle abstrakten Methoden implementieren.

Abstrakte Klasse versus Interface

- Abstrakte Klassen sind konzeptionell ähnlich zu Interfaces.
- Sie können jedoch nicht austauschbar eingesetzt werden:
  - Eine abgeleitete Klasse kann immer nur eine Basisklasse haben (auch wenn es eine abstrakte Klasse ist).
  - Eine Klasse kann jedoch beliebig viele Interfaces implementieren.

Abstrakte Klasse

- Zugriffsschutz: alle Stufen nach Bedarf.
- Objektvariablen: alle Arten erlaubt, auch private.
- Konstruktor: erlaubt, von abgeleiteten Klassen verwendet.

Interface

- kein Zugriffsschutz, alles ist public deklariert.
- Außer Methoden nur öffentliche Konstanten erlaubt.
- Kein Konstruktor erlaubt.
- Keine Implementierungen von Objektmethoden.

Redefinition (Override) von Methoden

- Abgeleitete Klassen können Methoden der Basisklasse neu definieren (**Redefinition**).
  - Die Signatur der Methode bleibt gleich in der abgeleiteten Klasse gleich.
  - Die Funktionalität der Methode wird neu definiert (überschrieben).
- Regeln zur Redefinition
  - **Name** und **Parameterliste** müssen exakt übernommen werden.
  - der **Zugriffsschutz** darf gleich bleiben oder gelockert werden, aber niemals eingeschränkt werden.
  - Der **Ergebnistyp** darf gleich bleiben oder muss kompatibel zum Ergebnistyp der redefinierten Methode sein (bei Referenztypen).
  - Der Rumpf kann komplett ersetzt werden. Das Verhalten der Basisklasse sollte dabei erhalten und ggf. erweitert werden.
  - Die Methode wird mit der Annotation **@Override** markiert. Hilft dem Compiler zu prüfen, ob die Regeln der Redefinition eingehalten wurden.
- Vorsicht: Eine abweichende Parameterliste bedeutet **Überladen** einer ererbten Methode, keine Redefinition!

Beispiel

```
public class X {
    int m(int x, int y) {
        return x * y;
    }
}

public class Y extends X {
    // redefiniert Methode von X
    @Override
    int m(int x, int y) {
        return x / y;
    }
}
```

```
X x1 = new X();
// Methode in X
x1.m(1, 2) -> 2

X x2 = new Y();
// redef. Methode in Y
x2.m(1, 2) -> 0

Y y = new Y();
// redef. Methode in Y
y.m(1, 2) -> 0
```

Folgen der Redefinition

- Abgeleitete Klassen können die Funktionalität der Basisklasse erweitern oder ändern, aber keinesfalls einschränken.
- Es existiert kein Sprachmittel zum Ausblenden ererbter Methoden oder Objektvariablen.
  - z.B. public -> private nicht erlaubt.
- Fazit: Ein abgeleitetes Objekt kann alles, was ein Basisklassenobjekt kann, möglicherweise etwas abgewandelt oder auch mehr, aber keinesfalls weniger.

Zugriff auf Methoden der Basisklasse

- Wird eine Methode in einer abgeleiteten Klasse redefiniert, ist die ursprüngliche Methode verdeckt.
- Der Bezug auf die Basisklasse bleibt über **super** erhalten.

```
doSomething()          // redefinierte Methode
super.doSomething()    // Methode in Basisklasse
```

- Hilfreich, wenn allgemeine Funktionalität der Basisklasse in den abgeleiteten Klassen genutzt und lediglich erweitert werden soll.

Statischer und dynamischer Typ

- Objekte abgeleiteter Klassen können
  - Variablen vom Typ der Basisklasse oder
  - Variablen vom Typ der abgeleiteten Klasse

  zugewiesen werden.
- Wie bei Interfaces Unterscheidung zwischen statischem Typ und dynamischem Typ:

```
// statischer und dynamischer Typ Laboratory
Laboratory l = new Laboratory(…);
// statischer Typ Room, dynamischer Typ Laboratory
Room r = new Laboratory(…);
```

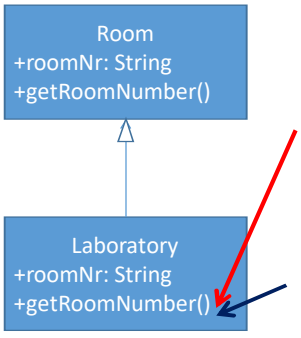

Kompatibilität

- Der dynamische Typ muss immer kompatibel zum statischen Typ sein.
- Ein Typ T ist kompatibel zu einem Typ U genau dann wenn einer Variable vom Typ U ein Wert vom Typ T zugewiesen werden kann.
  - Jeder Typ ist kompatibel zu sich selbst.
  - Ein primitiver Typ T ist kompatibel zu einem primitiven Typ U, zu dem es eine implizite Typkonversion T -> U gibt.
  - Eine abgeleitete Klasse ist kompatibel zu allen direkten und indirekten Basisklassen.
  - Eine Klasse ist kompatibel zu jedem Interface, das sie implementiert.

Statisches und dynamisches Binden

- Dynamisches Binden: Zur Laufzeit wird der dynamische Typ und damit die tatsächliche auszuführende Methode ermittelt. Gilt für:
  -> Objektmethoden
- Statisches Binden: Der Compiler legt zur Compile-Zeit fest, welche Implementierung / Definition verwendet wird. Gilt für:
  -> Konstruktoren
  -> statische Methoden, Klassenvariablen
  -> Objektvariablen

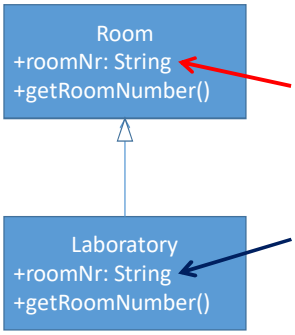
Dynamisches Binden von Objektmethoden




```
Room r = new Laboratory();
r.roomNumber;   // Objektvariable
r.getRoomNumber();   // Methode

Laboratory l = new Laboratory();
l.roomNumber;   // Objektvariable
l.getRoomNumber();   // Methode
```

Statisches Binden von Objektvariablen




```
Room r = new Laboratory();
r.roomNumber;   // Objektvariable
r.getRoomNumber();  // Methode

Laboratory l = new Laboratory();
l.roomNumber;   // Objektvariable
l.getRoomNumber();  // Methode
```


Zusammenfassung

- Klassen können von anderen Klassen Verhalten in Form von Methoden erben.
- Über den Aufruf super(…) im Konstruktor wird sichergestellt, dass immer auch der Basisklassen-Konstruktor aufgerufen wird und alle Werte initialisiert.
- Abgeleitete Klassen können Methoden der Basisklasse direkt erben und/oder redefinieren.
- Zur Laufzeit entscheidet sich welche konkrete Implementierung einer Methode tatsächlich ausgeführt wird (dynamisches Binden).
- Klassen können abstrakt sein werden. In diesem Fall können keine Objekte der abstrakten Klasse selbst initialisiert werden.
- Abstrakte Klassen können abstrakte Methoden definieren. Diese müssen in allen konkreten abgeleiteten Klassen implementiert werden.