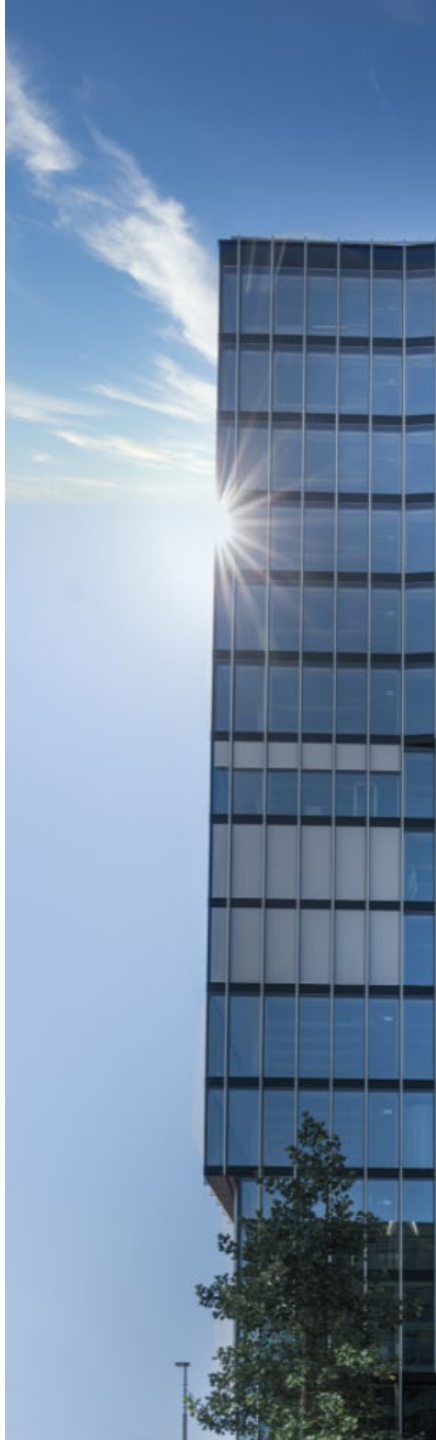
HSLU Hochschule Luzern

Objektorientierte Programmierung

Vererbung

Roland Gisler

FH Zentralschweiz



Inhalt

- Vererbung – Einführung
- Begriffe: Vererbung, Erweiterung, Ableitung, Spezialisierung
- Vererbung mit Klassen
- Vererbung mit Interfaces
- Einsatz der Vererbung – Hinweise und Empfehlungen
- Vererbung – ein einfaches Beispiel
- Zusammenfassung

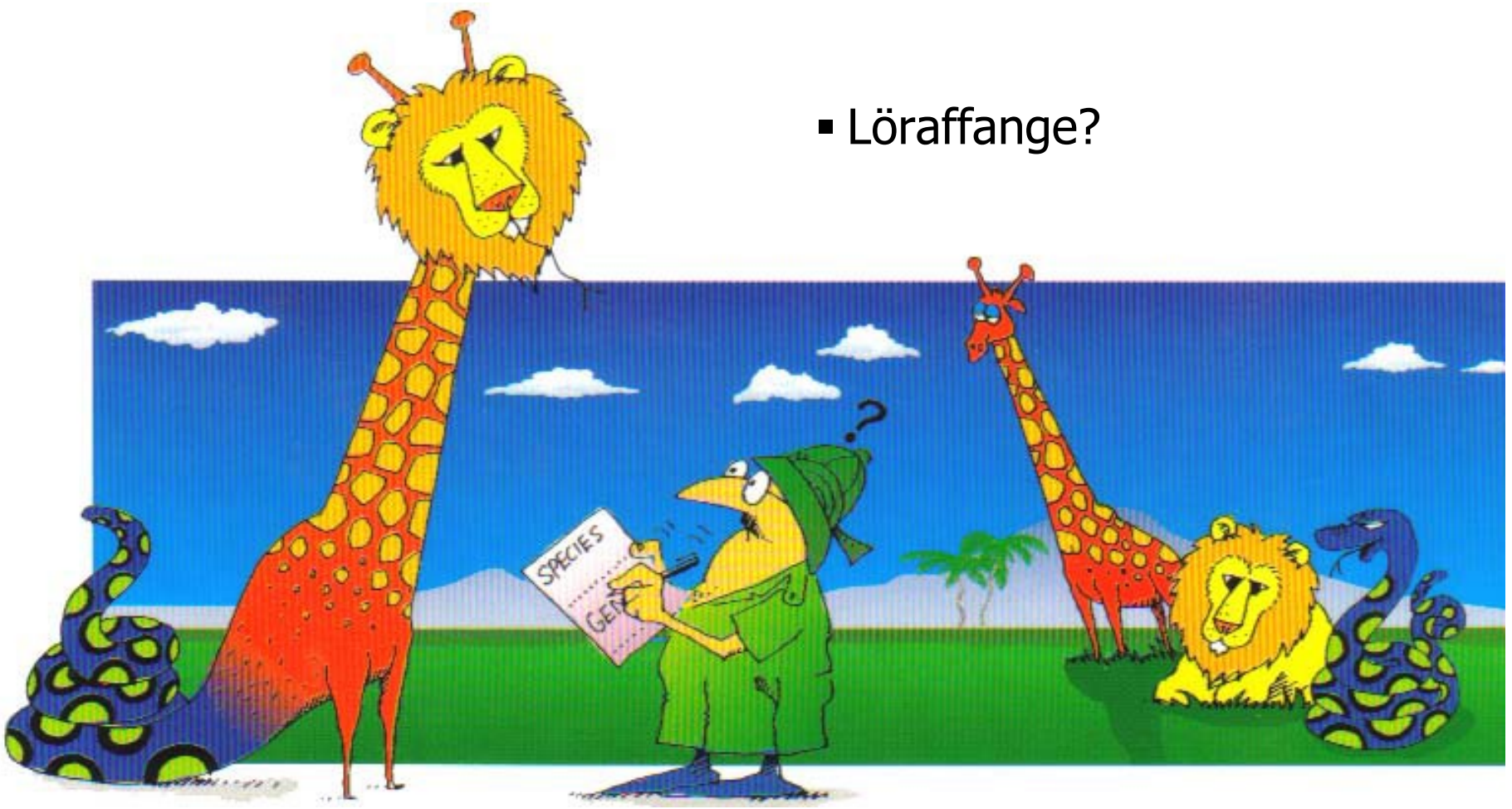
Lernziele

- Sie verstehen das Konzept der Vererbung.
- Sie können zwischen der Vererbung von Interfaces und Klassen unterscheiden.
- Sie kennen den Unterschied zwischen Einfach- und Mehrfachvererbung.
- Sie sind in der Lage, gute und schlechte Vererbungsbeziehungen zu identifizieren.
- Sie wissen, wie man Methoden in Spezialisierungen überschreiben und die Implementation der Basisklasse wiederverwenden kann.

Was ist Vererbung?

Vererbung in der Natur?

- Löraffange?



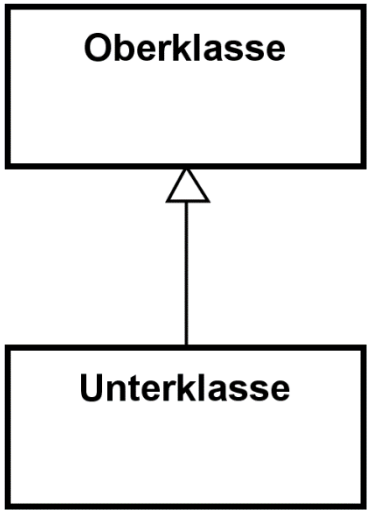
Bildquelle: Objektorientierte Analyse und Design; Grady Booch

Vererbung in der Natur vs. Objektorientierung

- Der Vergleich mit der Natur ist zwar naheliegend und amüsant, aber er vermittelt einen **völlig falschen Eindruck**!
- Vererbung in der OO folgt ganz **klaren** und **eindeutigen** Regeln.
  - Keine «zufällige» Vererbung von Eigenschaften!
- Viele hielten die Vererbung für eines der wichtigsten Konzepte der Objektorientierung, was es aber **nicht** ist!
  - Es gibt auch OO-Sprachen, die (sehr gut!) ohne Vererbung zu Recht kommen.
- Richtig eingesetzte Vererbung ist gut, schlecht eingesetzte Vererbung hingegen eine Katastrophe!
- Vererbung ist die **stärkste** Kopplung in der Objektorientierung.
  - Erinnerung: Wir wünschen uns möglichst lose Kopplung!

Vererbung - Begriffe

Vererbung – Begriffe

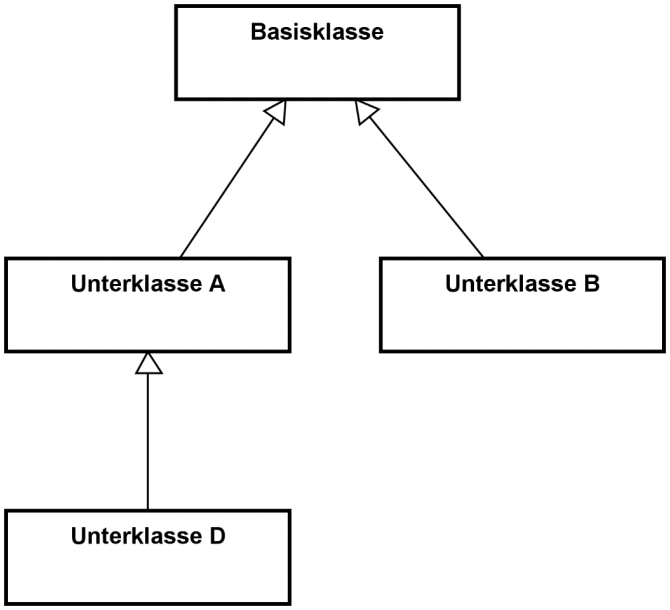


- Eine Unterklasse…
  - **erbt** von ihrer Oberklasse.
  - **erweitert** (extends) die Oberklasse.
  - ist eine **Ableitung** der Oberklasse.
  - **spezialisiert** die Oberklasse.
- Eine Oberklasse…
  - **vererbt** ihre Eigenschaften (Attribute und Methoden) an die Unterklasse.
  - **generalisiert** die Unterklasse.
- Synonyme für
  - Oberklasse: Basisklasse*, Superklasse, Generalisierung
  - Unterklasse: Subklasse, Spezialisierung, Ableitung, Subtyp

* Basisklasse bezeichnet die Basis einer Vererbungshierarchie (im jeweiligen Kontext).

Vererbungshierarchie - Vererbung über mehrere Stufen

- Es ist problemlos möglich, über mehrere Stufen zu vererben.
- Dadurch entstehen Vererbungshierachien bzw. -bäume*:



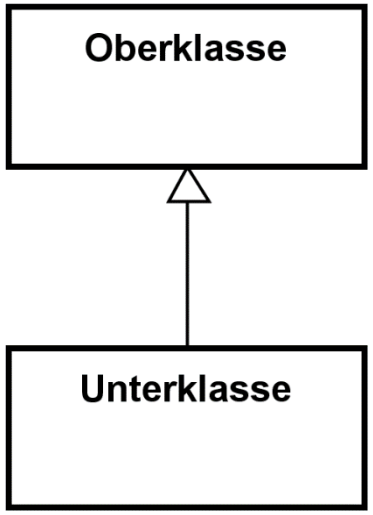
- Hinweis: Die Anzahl der Vererbungsstufen ist **kein Mass** für gutes Design, ganz im Gegenteil: Meistens ist weniger besser.

* in der Informatik **hängen** die Bäume meistens. ☺

Vererbung bei Klassen

Vererbung bei Klassen

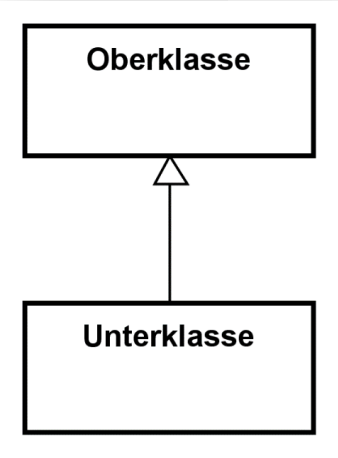
- Wenn eine Klasse von einer anderen Klasse erbt, dann
  - erbt sie **alle** Attribute der Oberklasse.
  - erbt sie **alle** Methoden (inkl. Konstruktoren) der Oberklasse.
- Die **Unterklasse** ist gleichzeitig auch vom Typ der Oberklasse ➔Subtyping, ➔Polymorphie.
- Bei **guter** Vererbung stimmt die folgende Aussage: Ein Objekt der Unterklasse **ist** auch **ein** Objekt der Oberklasse (**is-a**) und kann somit an dessen Stelle treten. Einfache Beispiele:
  - «Ein Apfel ist ein Kernobst.»
  - «Eine Student*in ist eine Person.»



Vererbung bei Klassen – Java Syntax

- Schlüsselwort: **`extends`** (im Klassenkopf)

```
public class Unterklasse extends Oberklasse {
    ...
}
```



- Eine Klasse kann nur von **einer** Oberklasse erben.
  - Konzept der Einfachvererbung.
- Ohne explizite Angabe einer Oberklasse erbt jede Klasse implizit von der Klasse **`Object`**.
- Somit ist **`Object`** die **Basisklasse** jeder Klasse, und jede Klasse bzw. jedes Objekt in Java ist ein (is-a) **`Object`**.
  - Dieser Umstand hat sehr viele (positive) Konsequenzen, welche wir später noch kennen lernen werden.

Mehrfachvererbung von Klassen - Java

- In Java hat jede Klasse genau **eine** Oberklasse.
  - Java kennt bei Klassen **keine** Mehrfachvererbung!
- Das ist **kein** Nachteil, da Java **➔Interfaces** kennt.
- Mehrfachvererbung wäre allenfalls im Zusammenhang mit vollabstrakten Klassen sinnvoll.
  - Vollabstrakte Klassen ersetzen in manchen Sprachen (z.B. C++) die fehlenden Interfaces, die Java aber beherrscht.
- «Echte» Mehrfachvererbung benötigt man nur sehr selten, und sie kann sehr problematisch sein!
  - Welche Methode wird verwendet, wenn mehrere Oberklassen (verschiedene) Implementationen anbieten?
  - Vergleiche: «**Anti**affen» und «Gir**lopen**».

Vererbung und Datenkapselung

- Attribute und Methoden von **Oberklassen** sind für die Unterklassen abhängig vom jeweiligen ➔**Zugriffsmodifikator** sichtbar oder nicht:
  - **`private`** – nicht sichtbar (aber trotzdem vorhanden!).
  - *<package default>* – sichtbar, wenn im selben Package.
  - **`protected`** – sichtbar, egal in welchem Package.
  - **`public`** – sichtbar für alle Klassen.
- Hinweis: Auch in Vererbungsbeziehungen zieht man private Attribute vor, und stellt diese wenn nötig über **`protected`**-Zugriffsmethoden den Spezialisierungen zur Verfügung!
  - Siehe dazu auch Input ➔Datenkapselung.

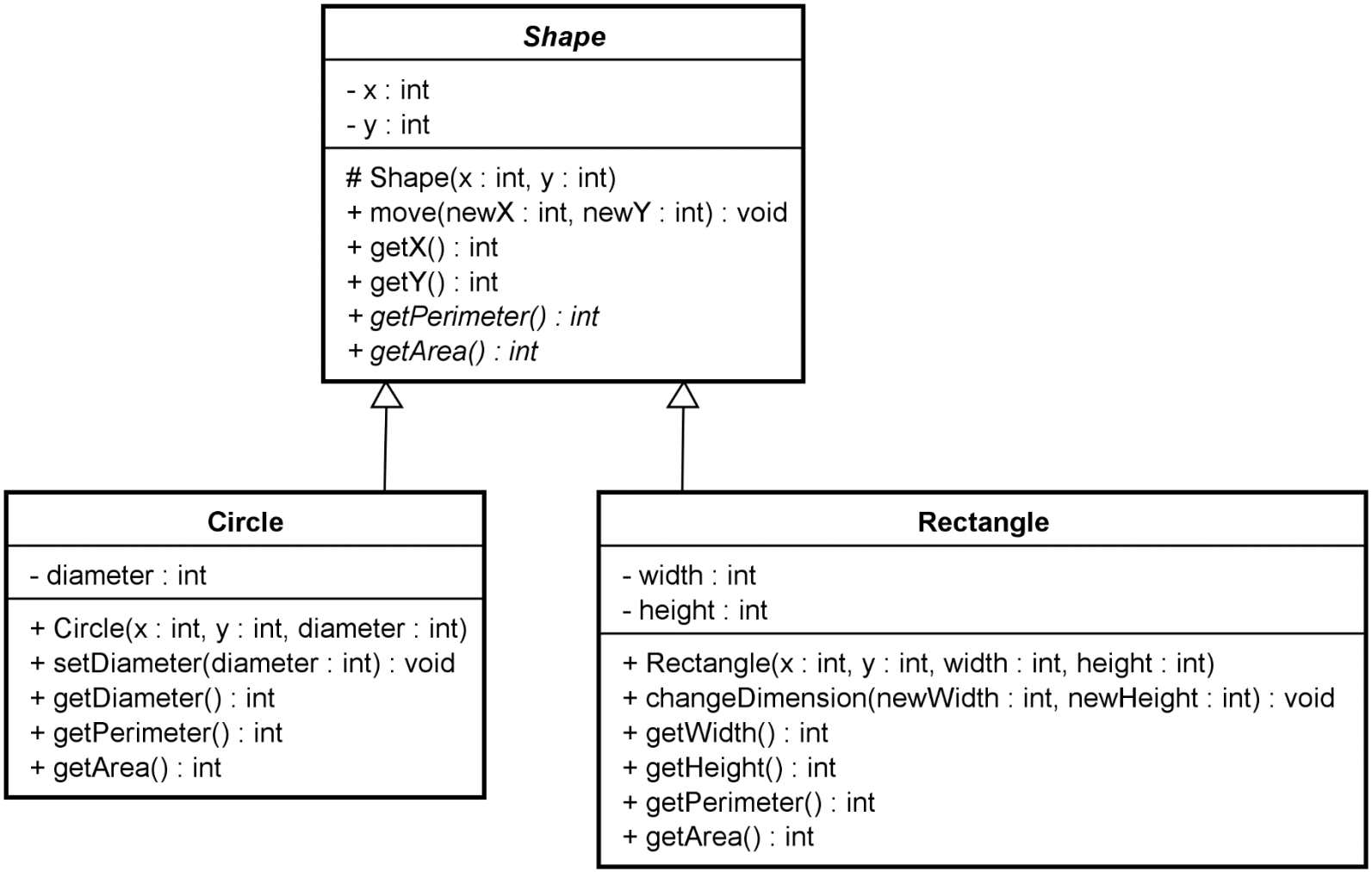
Vererbung und Konstruktoren

- Konstruktoren der Oberklasse werden ebenfalls vererbt.
- Sobald man aber in einer Unterklasse einen **eigenen** Konstruktor implementiert, **verdeckt** dieser **alle** Konstruktoren sämtlicher Oberklassen in der Vererbungshierarchie!
- Konstruktoren der Oberklasse können aber mit dem Schlüsselwort `super(…)` wieder explizit aufgerufen werden. Beispiel:

```
public Demo(…) {
    super(…);
    ...
}
```

- `super()`-Aufruf muss als **erstes** Statement im Konstruktor stehen!
  - Fehlt er, wird er vom Compiler automatisch eingefügt (sofern ein Standardkonstruktor vorhanden ist; ansonsten Fehler).

Vererbung von Klassen – Ein einfaches Beispiel



Einfaches Beispiel – Codefragment der Basisklasse

```
public abstract class Shape {

    private int x;
    private int y;

    protected Shape(final int x, final int y) {
        this.x = x;
        this.y = y;
    }

    public final void move(final int newX, final int newY) {
        this.x = newX;
        this.y = newY;
    }

    public abstract int getPerimeter();
    ...
}
```

Einfaches Beispiel – Codefragmente der Spezialisierung

```
public final class Rectangle extends Shape {

    private int width;
    private int height;

    public Rectangle(final int x, final int y,
                     final int width, final int height) {
        super(x, y);
        this.width = width;
        this.height = height;
    }
    ...

    @Override
    public int getPerimeter() {
        return (2 * this.width) + (2 * this.height);
    }
    ...
}
```

Einfaches Beispiel – Erklärung

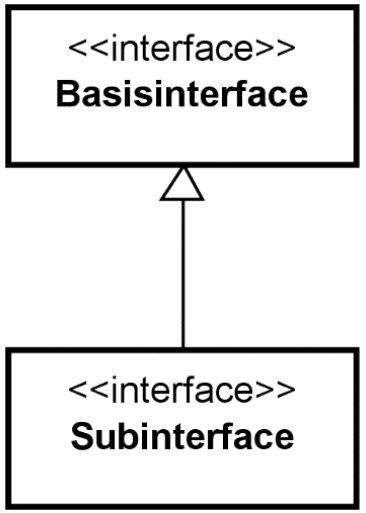
- Klasse ***Shape*** (abstrakt)
  - Definiert eine abstrakte Form mit einer (**x**,**y**)-Position.
  - Verlangt Methoden zur Berechnung von Umfang und Fläche.
- Klassen **`Circle`** und **`Rectangle`**
  - Spezialisieren die Klasse ***Shape***.
  - Implementieren individuell die abstrakten Methoden.
  - Ergänzen eigene, spezifische Attribute und Methoden.
  - Implementieren eigene, spezifische Konstruktoren.
- Das ist ein Beispiel für eine gute Vererbung.
  - Aber Vorsicht: **Nur im aktuellen** Funktionsumfang!
  - Sobald zusätzliche Anforderungen dazu kommen, kann eine bestehende Vererbung plötzlich nicht mehr geeignet sein!
  - Klassiker: Was passiert, wenn ***Shape*** plötzlich skalieren soll?

Vererbung von Interfaces

Vererbung bei Interfaces

- Wenn ein Interface von einem anderen Interface erbt, dann erbt es alle (abstrakten) Methoden des Oberinterfaces.
- Das Subinterface ist gleichzeitig auch vom Typ des Basisinterface ➔**Subtyping** und ➔**Polymorphie**.
- Bei guter Vererbung stimmt die folgende Aussage: Das Subinterface **ist ein** Basisinterface (**is-a**).

➔ Somit alles **analog** zur Vererbung bei Klassen.



Vererbung bei Interfaces – Java Syntax

- Schlüsselwort: **`extends`** (im Interfacekopf)

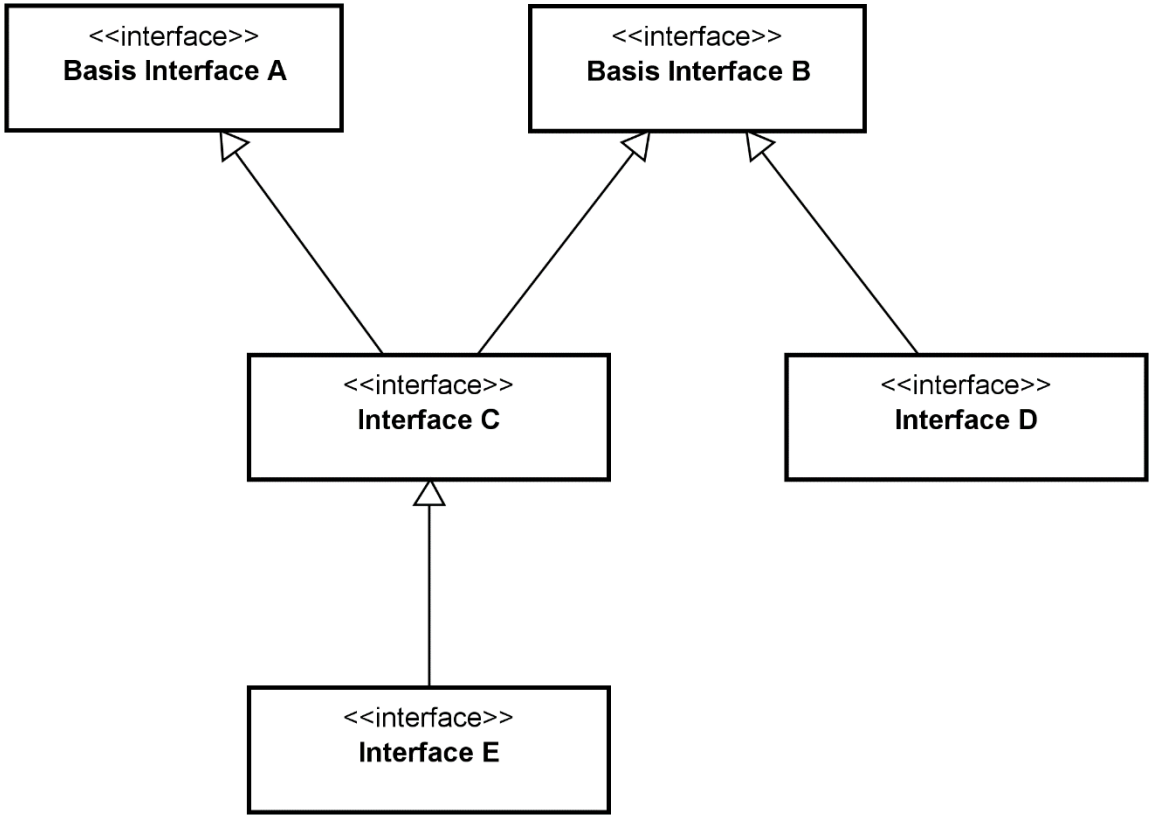
```
public interface Subinterface extends BasisInterface {
    ...
}
```

- Im Gegensatz zu Klassen kann ein Interface jedoch auch von **mehreren** Interfaces erben!
  - Keine Mehrdeutigkeit möglich, da in Interfaces ja keine Implementationen vorhanden sind.
  - Aber: Die ➔**Signaturen** müssen **einzigartig** sein.
- Mehrere Basisinterfaces können mit Komma separiert angegeben werden. Beispiel:

```
public interface Subinterface extends Interf1, Interf2 {
    ...
}
```

Mehrfachvererbung von Interfaces- Java

- Mehrfachvererbung **von Interfaces** ist bei Java möglich:



Einsatz der Vererbung

Vererbung - Eigenschaften

- Vererbung erlaubt es, eine Klasse als Erweiterung (Spezialisierung) einer anderen Klasse zu definieren.
- Alle gemeinsamen Eigenschaften (Attribute und Methoden) können in einer gemeinsamen Basisklasse implementiert werden.
  - Die abgeleiteten Unterklassen erben die Eigenschaften der Basisklasse und können diese verwenden.
  - Die Unterklassen brauchen nur noch die spezifischen Eigenschaften zu implementieren ➔Spezialisierung.
- Spezialisieren heisst, dass wir in der Unterklasse bestehende Methoden überschreiben oder neue Methoden ergänzen.
- **Achtung:** Vererbung verursacht eine sehr **starke Kopplung** und sollte **zurückhaltend** und **wohlüberlegt** eingesetzt werden!

Kriterien für gute Vererbung

- Eine Vererbung ist nur dann passend, wenn Sie einen «**is-a**» Satz bilden können. Beispiel: «*Ein Student ist eine Person.*»
- Widerstehen Sie insbesondere der Versuchung der Vererbung, wenn zwei Klassen (zufällig) sehr ähnliche/identische Attribute und/oder Methoden haben!
  - Beispiel: Kreis und Ellipse sind sich zwar sehr ähnlich (der Kreis ist sogar ein «Spezialfall» einer Ellipse), trotzdem darf man **nicht** sagen, dass ein Kreis eine Ellipse **ist**!
- Im Zweifelsfall besser auf Vererbung verzichten und stattdessen eine Komposition machen:
  - ➔ **F**avor **C**omposition **o**ver **I**nheritance (FCoI)
  - Beispiel: Ein Auto **hat** einen Motor. (Nicht: ...ist ein Motor.)

Vererbung - explizit erlauben oder aktiv verhindern!

- Wenn man beliebige Klassen spezialisieren kann (insbesondere auch solche, die nie dafür vorgesehen waren), können sich daraus vielfältige Probleme ergeben.
- Daraus hat sich eine interessante Designregel entwickelt: «Entweder entwirft man Klassen explizit zur Spezialisierung, oder aber man verhindert diese aktiv!»
- Schlüsselwort `final` **verhindert** die Spezialisierung einer Klasse:

```
public final class Klasse { … }
```

- Mindestens sollte man aber einzelne Methoden explizit vor dem ➔Überschreiben schützen (konkret: **alle** die nicht abstrakt sind):

```
…
    public final void foo() { … }
…
```

Vererbung – Empfehlungen und Hinweise

- Tendenziell folgt eine Vererbungshierarchie **gutem** Design, wenn:
  - Die Spezialisierung immer und überall den Platz der Basisklasse einnehmen könnte!
  - Die Basisklasse [voll-]abstrakt ist.
  - Die Methoden, die in der Spezialisierung implementiert werden, in der Basisklasse entweder leer oder abstrakt sind.
- Hinweise auf zweifelhaftes Design können umgekehrt sein:
  - Spezialisierung erweitert (oder verändert vollständig) die Implementation bestehender Methoden der Basisklasse.
  - Spezialisierung verbietet die Aufrufe (mit explizitem Fehler) von ausgewählten Methoden der Basisklasse.
  - Spezialisierung implementiert Methoden in Abhängigkeit vom effektiven Laufzeittyp des Objektes.

Zusammenfassung

- Vererbung ist ein sehr **mächtiges Konzept**, das **wohlüberlegt** eingesetzt werden sollte.
- Objektorientierung setzt nicht zwingend Vererbung voraus.
  - Vor allem in Sprachen, die Interfaces kennen.
- Im Zweifelsfall lässt sich jede Vererbung auf eine einfache Komposition zurückführen (also nicht **is-a**, sondern **has-a**).
  - ggf. mit Interface implementieren (eine Rolle übernehmen)
- Potenzielle Basisklassen sollten als solche identifiziert und auch ganz bewusst entsprechend implementiert werden.
  - abstrakte Basisklassen und/oder Methoden.
- Ist eine Klasse nicht für Spezialisierung vorgesehen, sollte man diese durch Finalisierung (`final`) explizit verhindern.

Fragen?