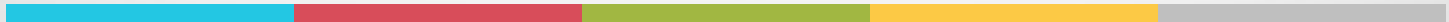
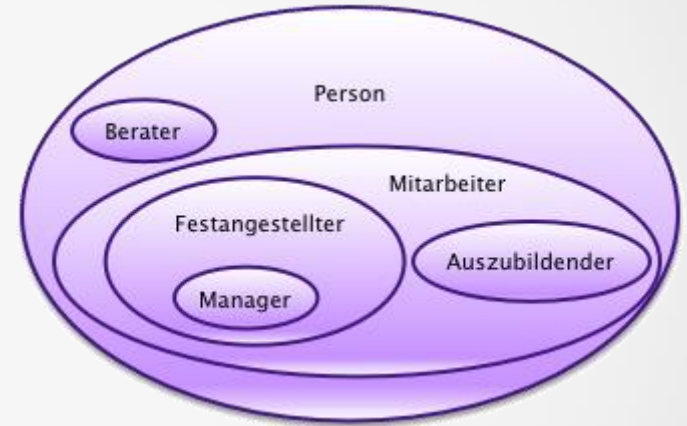
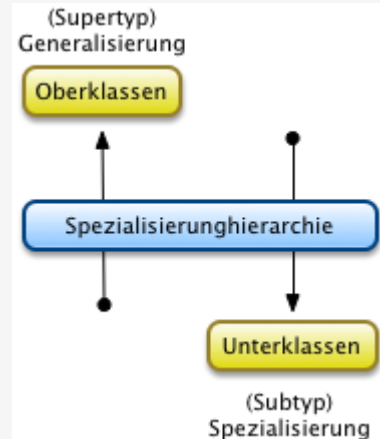


OOP – Vererbung



Motivation zur Vererbung von Klassen



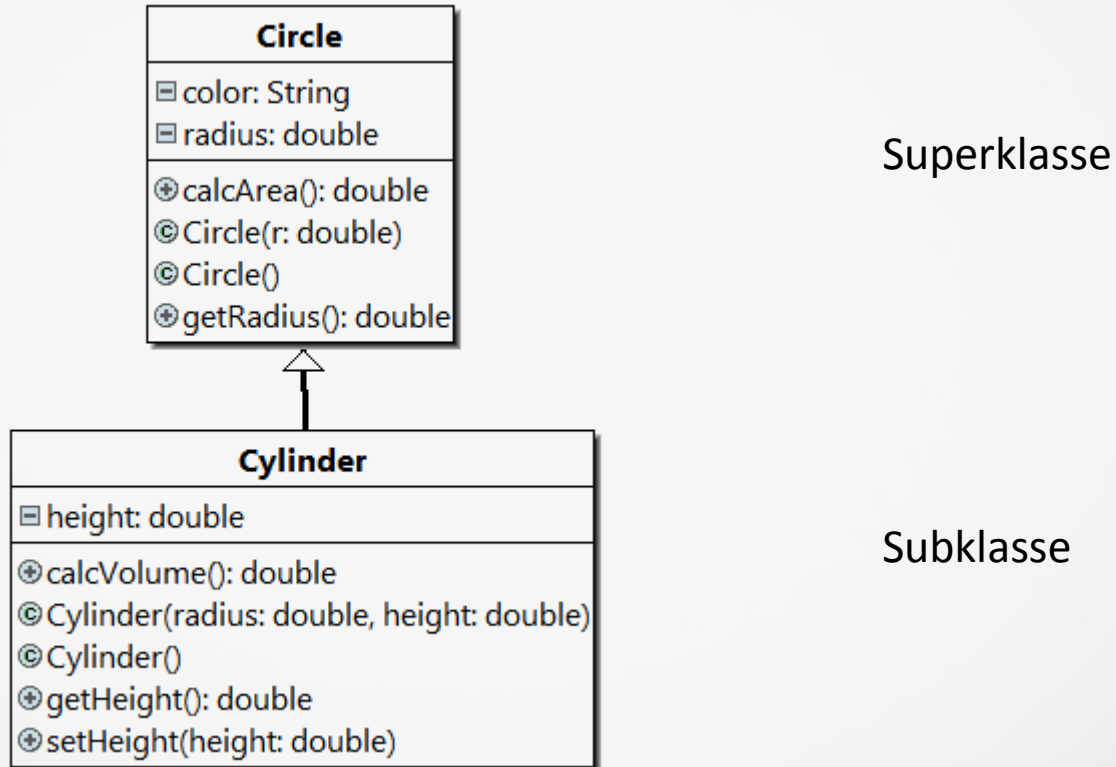
Allgemeine Regeln I

- ✓ Modellierung einer **"ist-ein"**-Beziehung zwischen Datentypen
- ✓ manchmal ist ein Datentyp A ein **Untertyp** eines **Obertyps** B
- ✓ die Klasse A ist dann eine **Erweiterung** der Klasse B,
d.h. alles was B hat hat auch A
- ✓ Syntax: **class A extends B**
- ✓ alle Methoden von B sind somit in A und über **A-Objekte** verwendbar
- ✓ A kann Methoden von B durch eigene (passendere) Methoden **überschreiben**

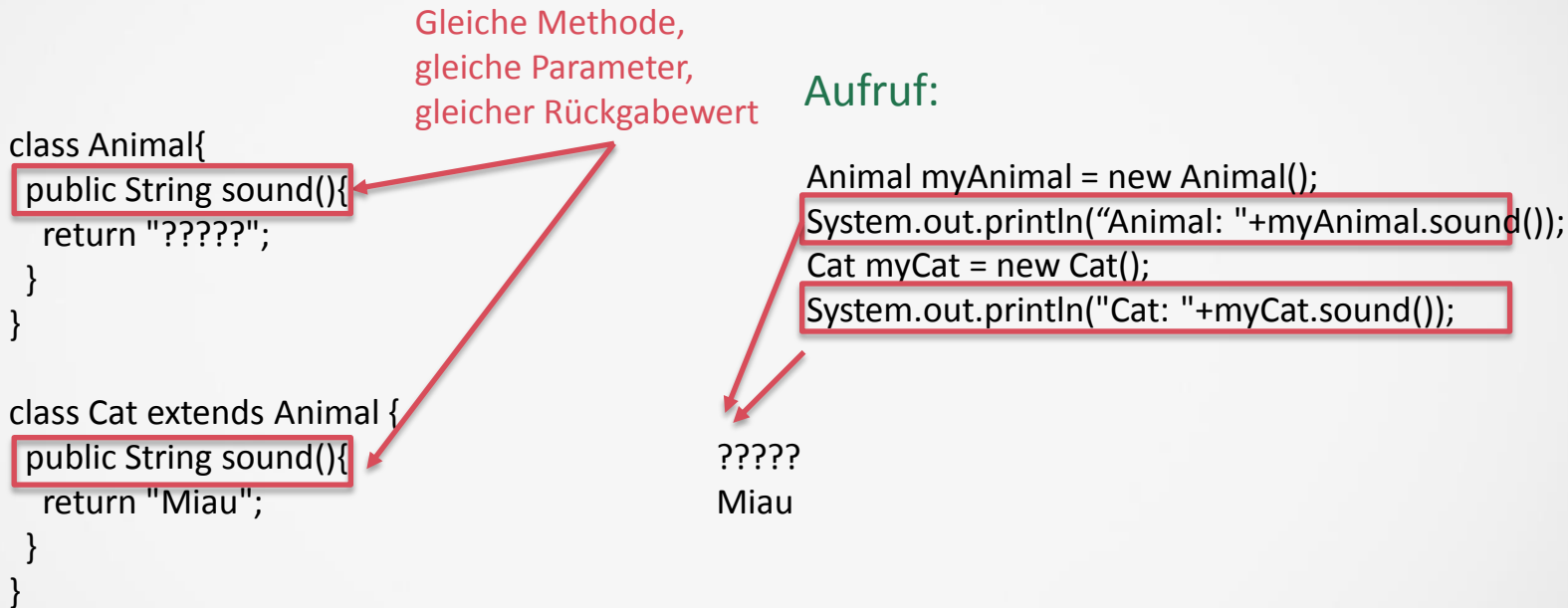
Allgemeine Regeln II

- ✓ Von **einer** Klasse B können **beliebig viele** Klassen abgeleitet werden
- ✓ eine Klasse A kann nur von **einer** Klasse erben (**single inheritance**)
- ✓ in einem Konstruktor von A wird implizit der argumentlose Konstruktor von B aufgerufen **super()**
- ✓ andere Konstruktoren können explizit mit **super(...)** (als erste Anweisung) angesprochen werden
- ✓ durch endgültige Klassen (**final** class) lässt sich die Vererbung / Ableitung verhindern
- ✓ **Wiederverwendbarkeit**: der Code der Basisklasse muss nicht neu programmiert werden

Beispiel Cylinder->Circle I



Überschreiben/Überladen von Methoden I



Nur die Sichtbarkeit der Methode darf erhöht werden: protected->public

Beispiel Cylinder->Circle II – Elternklasse Circle

Die Methoden
setRadius, getColor
und setColor wurden
auf Grund der
Übersichtlichkeit
weggelassen!

Attribute

```
public class Circle {  
    private double radius;  
    private String color;  
}
```

1. Konstruktor

```
public Circle() {  
    radius = 1.0;  
    color = "red";  
}
```

2. Konstruktor

```
public Circle(double r) {radius = r;  
    color = "red";  
}
```

Getter für radius

```
public double getRadius() {  
    return radius;  
}
```

Methode

```
public double calcArea() {  
    return radius*radius*Math.PI;  
}
```

```
}
```

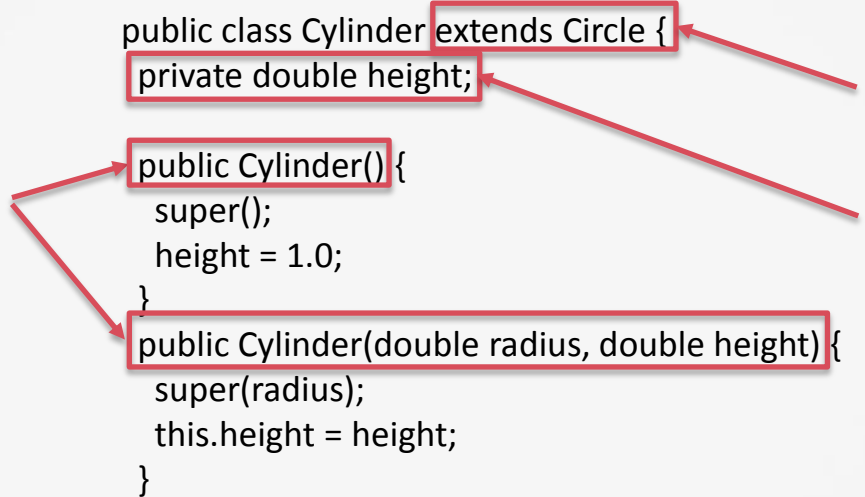
Beispiel Cylinder->Circle II – Kindklasse Cylinder

Aufruf des
Konstruktors der
übergeordneten
Klasse

```
public class Cylinder extends Circle {  
    private double height;  
    public Cylinder() {  
        super();  
        height = 1.0;  
    }  
    public Cylinder(double radius, double height) {  
        super(radius);  
        this.height = height;  
    }  
}
```

Klassenname, von der
sich die Klasse ableitet

Neues Attribut



Beispiel Cylinder->Circle II – Kindklasse Cylinder

Setter und Getter
des neuen Attributes

```
public double getHeight() {  
    return height;  
}  
public void setHeight(double height) {  
    this.height = height;  
}
```

Zusätzliche neue
Methode der
abgeleiteten Klasse

```
public double calcVolume() {  
    return calcArea()*height;  
}  
}
```

Beispiel Cylinder->Circle IV – Hauptprogramm

```
public class TestCylinder {  
    public static void main(String[] args) {  
        Cylinder cylinder = new Cylinder();  
        //Cylinder cylinder = new Cylinder(5.0, 2.0);  
        System.out.println("Radius is " + cylinder.getRadius()  
            + " Height is " + cylinder.getHeight()  
            + " Full area is " + cylinder.getArea()  
            + " Volume is " + cylinder.getVolume());  
    }  
}
```

Zweiter möglicher Konstruktor

Methode der
ursprünglichen Klasse

Methode der
abgeleiteten Klasse

Beispiel Cylinder->Circle IV

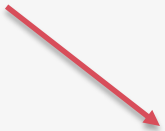
Überschreiben der Methode der alten Klasse

```
public double getArea() {  
    return 2*Math.PI*getRadius()*height + 2*super.getArea();  
}  
  
public double getVolume() {  
    return super.getArea()*height;  
}
```

Verwenden der Methode der Klasse Circle durch „super.“
(= Referenz auf die Superklasse)

“Super” |

Ist kein Konstruktor definiert, dann wird automatisch ein Konstruktor OHNE Parameter erzeugt:



```
public CCylinder() {  
    super();  
}
```

“Super” II

- ✓ Konstruktoren werden nicht vererbt.
- ✓ Konstruktoren der Basisklasse können mit `super(...)` aufgerufen werden.
- ✓ Ein solcher Aufruf von `super(...)` muss die erste Zeile der Konstruktordefinition sein

Motivation Vererbung

- ✓ Es ist immer gut, mit der Typstruktur des Programms dicht an der realen Welt zu bleiben (Leichteres Verständnis)
- ✓ Man spart Tippaufwand
- ✓ Damit auch den Aufwand, das Programm lesen und verstehen zu müssen
- ✓ Wenn man was für alle Objekte ändern möchte, muss man die Änderung nur bei dem Eltern Objekt vornehmen

Private, Protected und Public I

Zugriff erlaubt?

	Eigene Klasse	Elternklasse	Fremde Klasse = Tester(main)
private	ja		
protected	ja	ja	
public	ja	ja	ja

Private, Protected und Public II

```
public class CParent {
    public int ValuePub;
    protected int ValueProt;
    private int ValuePriv;
}

public class CChild extends CParent{
    public void TestPropAndMeths () {
        ValuePub=1;
        ValueProt=1;
        //ValuePriv=1;
    };
}
```



Fehlermeldung: „ValuePriv has private access in Cparent“

Regeln für Konstruktoren

```
class A {  
    A(int i) {...}  
}  
class B extends A {  
    int j;  
}
```

Fehler, da keine Konstruktor mit einem Parameter B(int i) vorhanden ist und kein Konstruktor B() impliziert:
error: constructor A in class A cannot be applied to given types;

Lösung:

```
class B extends A {  
    B(.....){  
        super(-1000);  
    };  
}
```

Beliebige Anzahl
und Typ von
Parametern

Keine Lösung:

```
class B extends A {  
    B(int i){  
    };  
}
```

Problem: Der Aufruf
von super erfolgt
immer! In diesem
Fall dann als Default
mit 0 Parametern

Klasse als abgeleitete Klasse von Object

```
class A {  
}
```

```
class A extends Object{  
}
```

Jede Klasse ist eine Kindklasse von Object!
Daher ist obiger Code gleichwertig!

Namenskonventionen

- ✓ Klassen: UpperCamelCase
- ✓ Methoden, Variablen: lowerCamelCase