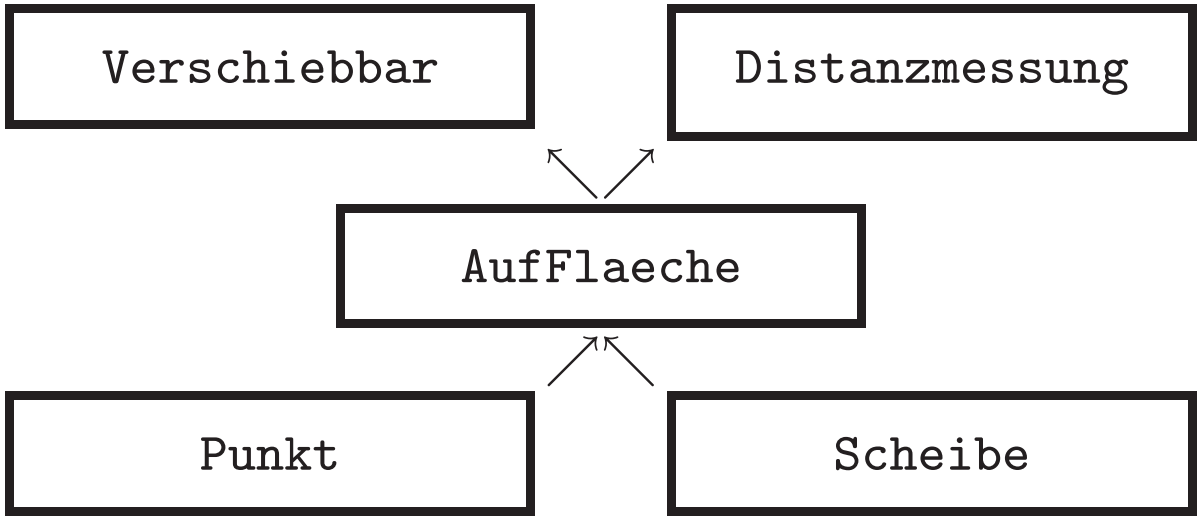
# Enthaltender Polymorphismus

Untertypbeziehungen:




Eine Instanz eines *Untertyps* ist überall verwendbar,
wo eine Instanz eines *Obertyps* erwartet wird.

```
public class FlaechenTester {
    private static void bewege(Verschiebbar v) {
        v.verschiebe(1.5,2.5);
    }
    private static void gibDistanzAus(Distanzmessung d) {
        System.out.println(d.distanzZumUrsprung());
    }
    public static void teste(AufFlaeche f) {
        for (int i = 0; i < 5; i++) {
            bewege(f);
            gibDistanzAus(f);
        }
    }
    public static void main(String[] args) {
        teste(new Punkt(0.0, 0.0));
        teste(new Scheibe(0.0, 0.0, 2.1));
    }
}
```

# Dynamisches Binden

- Variable hat gleichzeitig zwei Typen (wenn Referenz):
  - deklarierter Typ (steht in der Variablendeklaration)
  - dynamischer Typ (= Klasse, von der das Objekt in der Variablen erzeugt wurde; ändert sich bei Zuweisungen)
- dynamischer Typ bestimmt, durch Ausführung welcher Methoden Nachrichten beantwortet werden
- dynamisches Binden: dynamischer Typ möglicherweise ungleich deklariertem (= aufgerufene Methode zur Laufzeit)
- statisches Binden: dynamischer Typ immer gleich deklariertem (= Compiler kennt aufgerufene Methode)

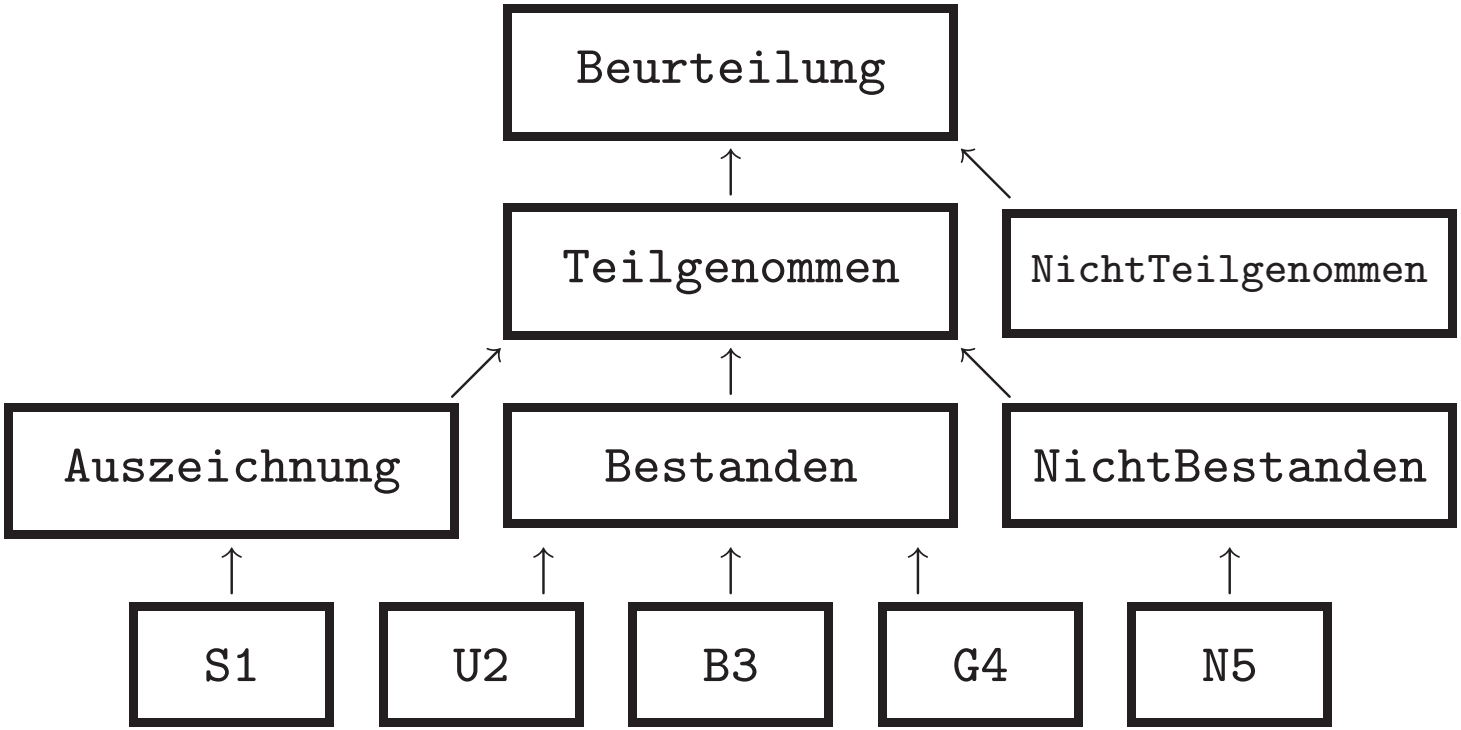
```
public interface Beurteilung {
    boolean bestanden();
    String toString();
}

public class Auszeichnung implements Beurteilung {
    public boolean bestanden() { return true; }
    public String toString() {
        return "mit Auszeichnung bestanden";
    }
}

public class Bestanden implements Beurteilung {
    public boolean bestanden() { return true; }
    public String toString() {
        return "bestanden";
    }
}
```

```
public class NichtBestanden implements Beurteilung {
    public boolean bestanden() { return false; }
    public String toString() {
        return "nicht bestanden";
    }
}

public class Test1 {
    public static String test (Beurteilung b) {
        String s = "Prüfung " + b.toString() + "! ";
        if (b.bestanden())
            return s + "Herzlichen Glückwunsch!";
        else
            return s + "Vielleicht das nächste Mal.";
    }
}
```

```
public class Test2 {
    public static final int AUSZ = 1, POS = 2, NEG = 5;

    public static String test (int b) {
        String s = "Prüfung ";
        String e = "Da ist etwas flasch gelaufen.";
        switch (b) {
            case AUSZ: s += "mit Auszeichnung ";
            case POS:  s += "bestanden";
                       e = "Herzlichen Glückwunsch!";
                       break;
            case NEG:  s += "nicht bestanden";
                       e = "Vielleicht das nächste Mal.";
                       break;
        }
        return s + "! " + e;
    }
}
```

# Spezialisierungen und die reale Welt

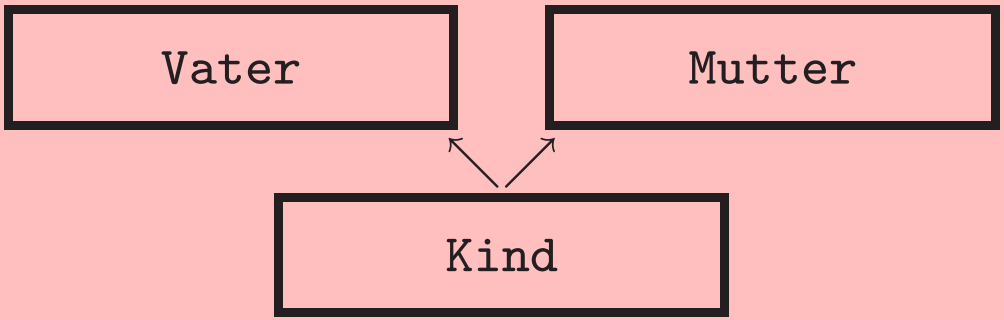
ist-ein-Beziehungen:



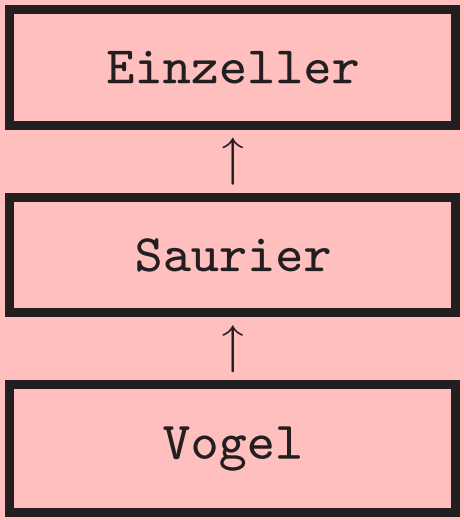
# Ersetzbarkeitsprinzip

*U ist Untertyp von T wenn Objekte vom Typ U überall verwendbar wo Objekte vom Typ T erwartet*

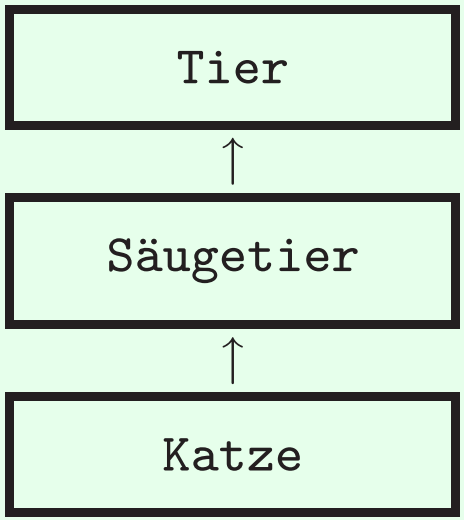
# Negatives Beispiel



# Negatives Beispiel



# Positives Beispiel



# Vererbung auf Klassen

```
public class Bestanden implements Beurteilung {
    public boolean bestanden() { return true; }
    public String toString() {
        return "bestanden";
    }
}

public class B3 extends Bestanden {   // bestanden() ererbt
    public String toString() {        // überschrieben
        return "mit \"befriedigend\" bestanden";
    }
    public int toInt() { return 3; }  // hinzugefügt
}
```

# Vererbung: Sichtbarkeit, Konstruktoren

```
public class FarbigeScheibe extends Scheibe {
    private long r;       // Farbe der Scheibe als RGB-Code

    public FarbigeScheibe (double initX, double initY,
                           double radius, long farbeRGB) {
        super(initX, initY, radius);
        r = farbeRGB;
    }

    public long farbe() { return r; }                    // OK

    // public double entfernung(Punkt p) {
    //     return (new Punkt(x,y)).entfernung(p);
    // }          // Syntaxfehler: x und y nicht sichtbar !!
}
```

# Modifier für Sichtbarkeit

**public** überall sichtbar

**private** nur innerhalb der Klasse sichtbar

**(default)** innerhalb des Paketes sichtbar

**protected** innerhalb des Paketes und in Unterklassen sichtbar

# Möglichkeiten der Vererbung

- nicht-statische Methoden und Variablen werden geerbt
- ererbte private Methoden und Variablen nicht sichtbar
- sichtbare ererbte Methoden überschreibbar
- Erweiterung um neue Methoden und Variablen
- gleichnamige Variablen verdeckt
- Nachricht senden $\rightarrow$ dynamisches Binden,
  Variablenzugriff $\rightarrow$ statisches Binden
- auch Konstruktoren der Oberklasse ausgeführt

# Klassen versus Interfaces

- jede Klasse von nur einer Klasse direkt abgeleitet
- jede Klasse kann mehrere Interfaces implementieren
- Interfaces können mehrere Interfaces erweitern
- nur public Methoden und Konstanten in Interfaces
- Code-Vererbung und Objekterzeugung nur auf Klassen

# Variationen: abstract, final

```
public abstract class Auszeichnung implements Beurteilung {
    public final boolean bestanden() { return true; }
    public String toString() {
        return "mit Auszeichnung bestanden";
    }
    public abstract int toInt();
}

public final class S1 extends Auszeichnung {
    public int toInt() { return 1; }
}

public class Fuenf extends Auszeichnung {
    public final int toInt() { return 5; }
}
```

# Klassen kombiniert mit Interfaces

```
public interface Note {
    int toInt();
}

public class B3 extends Bestanden implements Note { ... }

public abstract class Auszeichnung
                      implements Beurteilung, Note { ... }

public class Test {
    public static void test (Note n) {
        System.out.println(n.toInt());
    }
}
```