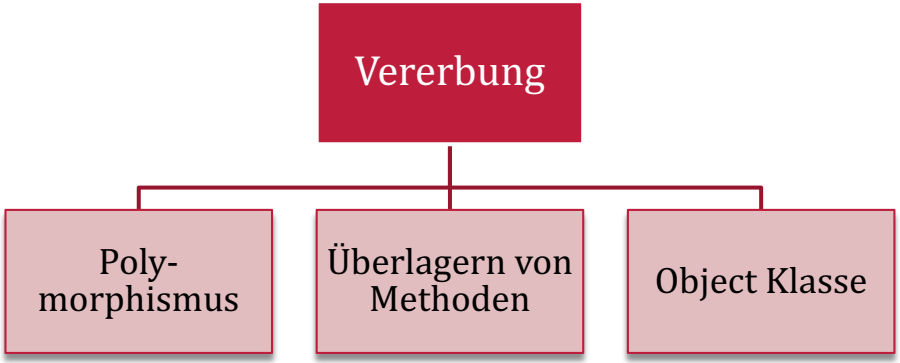
Technische Universität Braunschweig

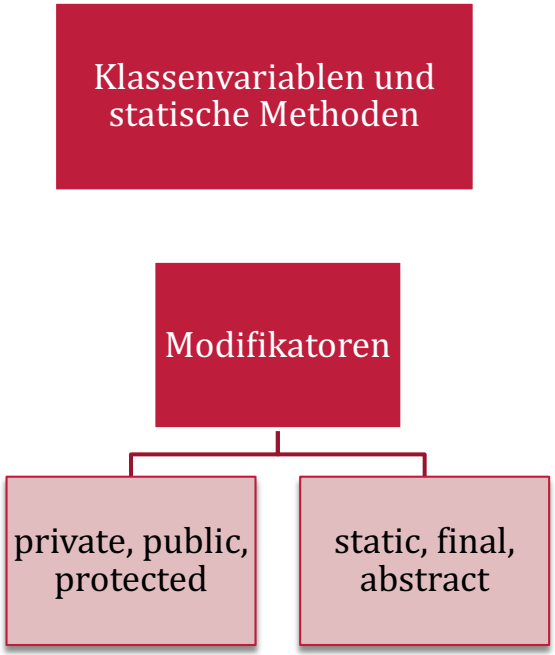
Programmieren 1 – Vorlesung #7

Arne Schmidt

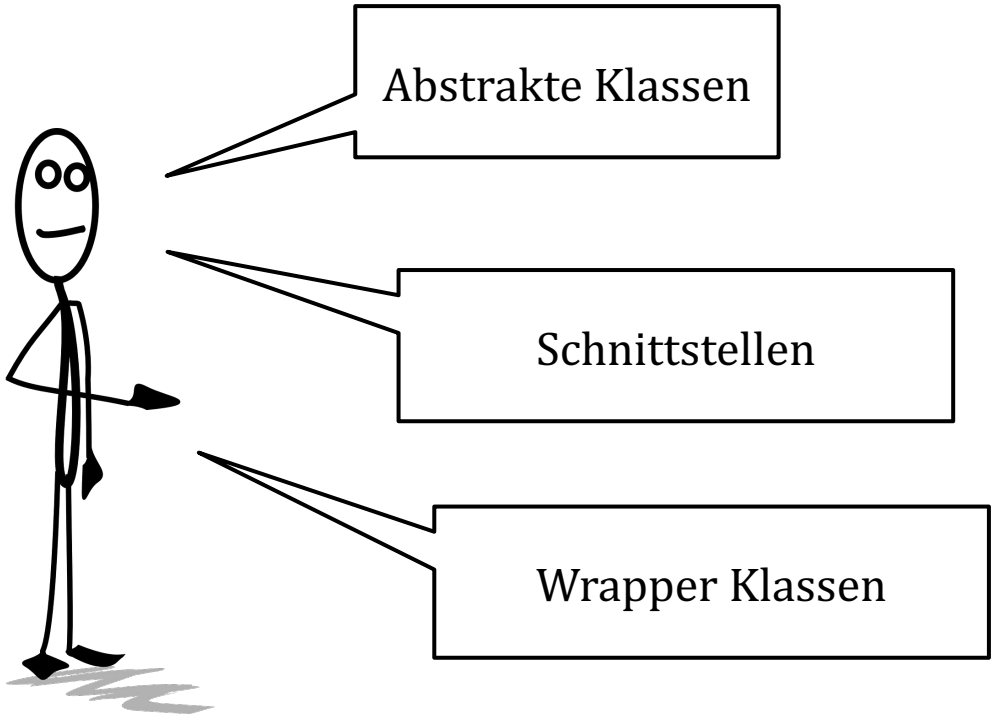
Wiederholung

Letzte Woche





Heute



Kapitel 4.5 – Abstrakte Klassen

Abstrakte Methoden

Eine Methode heißt **abstrakt**, wenn ihre Deklaration nur den Kopf (also Modifikatoren, Rückgabetyp, Bezeichner und Parameter), aber keine Implementierung (*Rumpf*) besitzt.

Konkrete Methoden sind in ihrer Deklaration vollständig.

Abstrakte Methoden führen den Modifikator **abstract** und schließen ihren Kopf durch ein Semikolon.

Beispiel:
abstract double gehalt( );

Abstrakte Methoden können nicht aufgerufen werden. Dies wird erst durch Überlagerung und einer Implementierung möglich.

Abstrakte Klassen

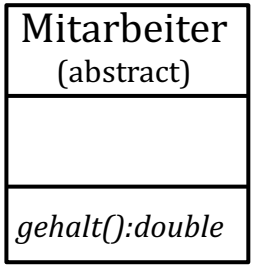
Wenn eine Klasse mindestens eine abstrakte Methode enthält, trägt sie den Modifikator **abstract**. Eine Klasse mit Modifikator abstract heißt **abstrakte Klasse**.

Da abstrakte Methoden nicht aufgerufen werden können, können abstrakte Klassen nicht instanziiert werden.

Klassen können von abstrakten Klassen erben und abstrakte Methoden implementieren.

Eine abstrakte Klasse kann ausschließlich aus konkreten Methoden bestehen.

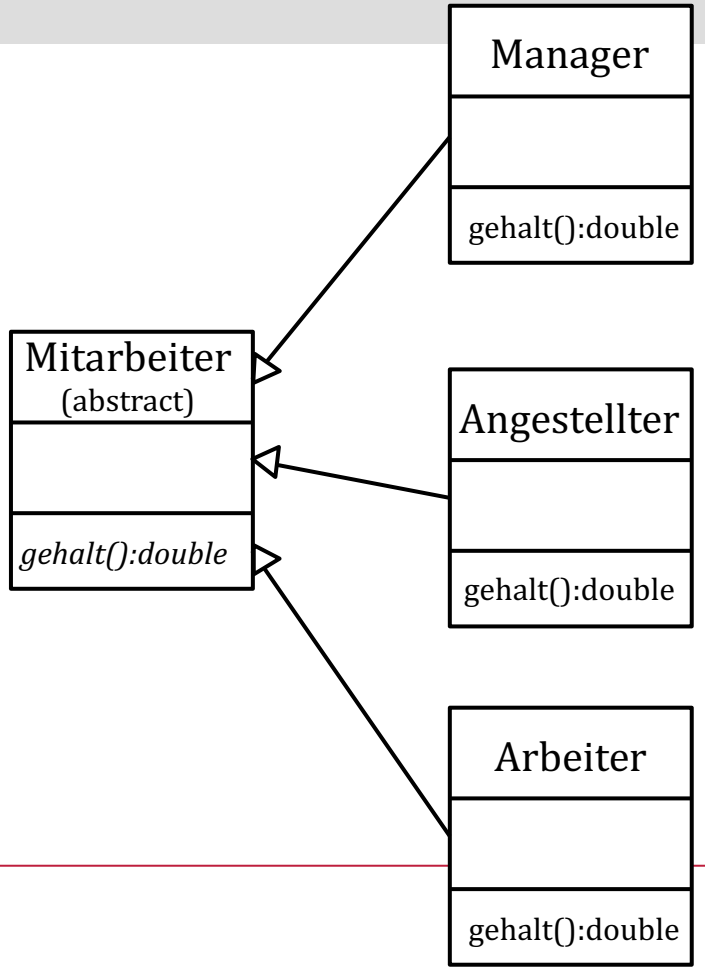
Im Klassendiagramm wird der Klassenname kursiv geschrieben oder unter dem Klassennamen die Eigenschaft “(abstract)” ergänzt.
Auch Methoden werden kursiv geschrieben oder mit (abstract) annotiert.



Abstrakte Klassen – Beispiel

In einer Firma gibt es drei Typen von Mitarbeitern: Manager, Angestellte und Arbeiter.

Alle bekommen ein Gehalt, welches aber unterschiedlich berechnet wird. Das muss konkretisiert werden!



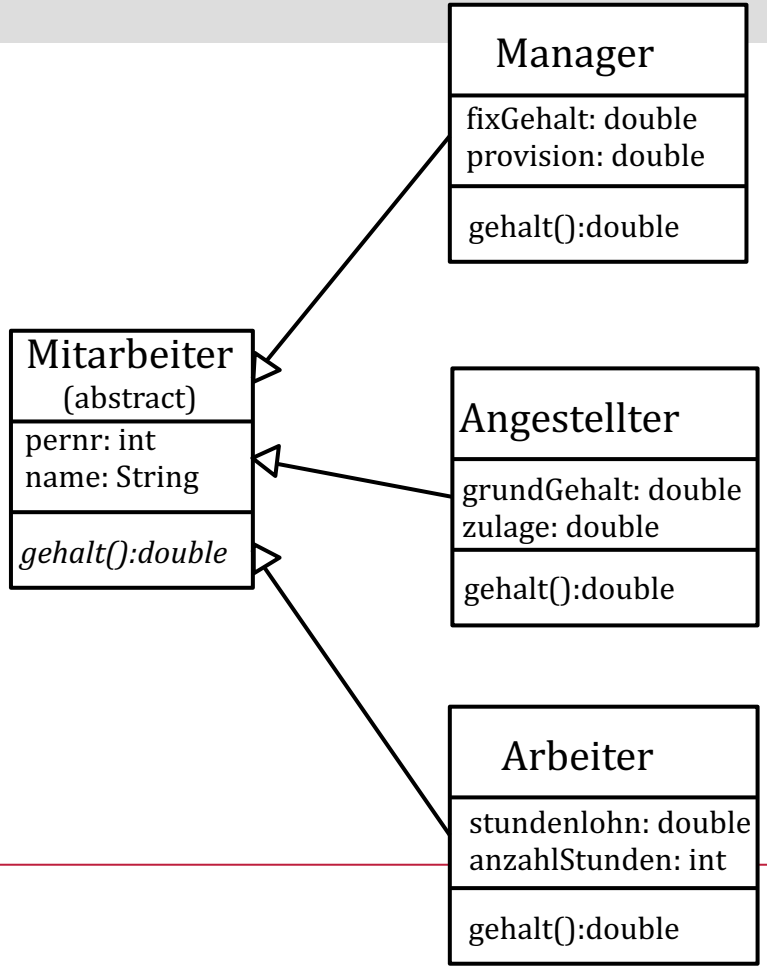
Abstrakte Klassen – Beispiel

In einer Firma gibt es drei Typen von Mitarbeitern: Manager, Angestellte und Arbeiter.

Alle bekommen ein Gehalt, welches aber unterschiedlich berechnet wird. Das muss konkretisiert werden!

Ausnutzen von Polymorphismus:

```
Mitarbeiter[] mitarb;
//... mitarb wird gefüllt!

for (Mitarbeiter pers : mitarb){
   System.out.println(pers.name + " erhält " +
         pers.gehalt() + "€ brutto");
}
```



Polymorphie in Konstruktoren

```
class SingleVal{
    int val1 = 1;

    public SingleVal(int val1){
        print();
        this.val1 = val1;
        print();
    }

    public void print(){
        System.out.println("S: " + val1);
    }
}
```

```
class DoubleVal extends SingleVal{
    int val2 = 2;

    public DoubleVal(int val1, int val2){
        super(val1);
        print();
        this.val2 = val2;
        print();
    }
    @Override
    public void print(){
        System.out.println("P: " + val1 + " " + val2);
    }
}
```

Welche Nachrichten werden mit der Anweisung new doubleVal(3, 4) ausgegeben?

Auflösung

P: 1 0

P: 3 0

P: 3 2

P: 3 4

Anmerkung:

val2 wird erst nach dem super()-Aufruf mit dem Wert 2 initialisiert
→ val2 besitzt in den ersten beiden print-Aufrufen den Wert 0.

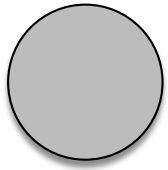
Polymorphe Methoden in Konstruktoren sollten mit Vorsicht genutzt werden!

Kapitel 4.6 – Schnittstellen

Schnittstelle

Eine abstrakte Klasse heißt **Schnittstelle** bzw. **Interface**, wenn sie nur abstrakte Methoden und Konstanten enthält.

Ein Interface wird mit dem Schlüsselwort interface anstelle von class definiert.

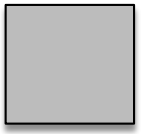



Alle Methoden eines Interfaces sind implizit abstract und public.

Konstanten sind final, static und public.

Ein Interface darf keine Konstruktoren enthalten.

```
interface Shape2D{
    double getArea( );
    double getBoundaryLength( );
    double getDiameter( );
    ...
}
```




Implementierung einer Schnittstelle

Um eine Schnittstelle zu implementieren, muss eine Klasse mit “implements Interfacename” angewiesen werden

Ein Interface kann von mehreren Klassen implementiert werden.

Eine Interface-Variable ist zu allen Klassen kompatibel, die dieses Interface implementieren.

Eine Klasse kann mehrere Interfaces implementieren!

```
class Circle implements Shape2D{
    double radius;

    public Circle(double radius){
        this.radius = radius;
    }

    @Override
    double getArea( ); {
        return radius * radius * Math.PI;
    }
    …
}
```

Implementierung einer Schnittstelle

@Override aus Platzgründen ausgelassen.

```
class Square implements Shape2D{
    double side_length;

    public Square(double length){
        this.side_length = length;
    }

    public double getArea( ) {
        return side_length * side_length;
    }

    public double getBoundaryLength( ) {
        return 4 * side_length;
    }

    public double getDiameter( ) {
        return side_length;
    }
}
```

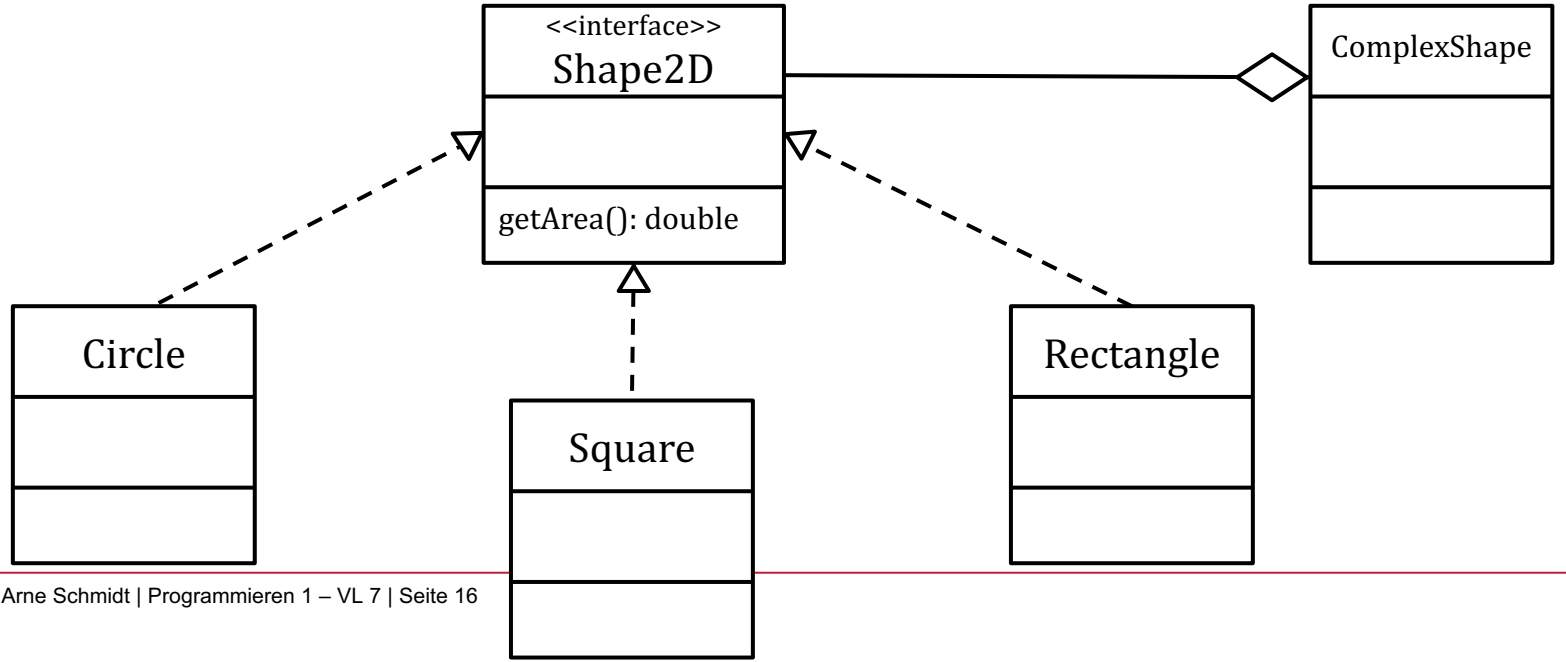
```
class Circle implements Shape2D{
    double radius;

    public Circle(double radius){
        this.radius = radius;
    }

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

    public double getBoundaryLength( ) {
        return 2 * radius * Math.PI;
    }

    public double getDiameter( ) {
        return 2*radius;
    }
}
```

```
Shape2D[] shapes = new Shape2D[5];

shapes[0] = new Square(5);
shapes[1] = new Circle(3.2);
shapes[2] = new Circle(3.2);
shapes[3] = new Square(2.9);
shapes[4] = new Square(10);

double totalArea = 0;
for (shape2D s : shapes){
   totalArea += s.getArea();
}

System.out.println("Total area is " +
      totalArea);
```

Ausgabe: *Total area is 197.749…*

Schnittstellen in Klassendiagrammen

Eine Schnittstelle im Klassendiagramm enthält den Stereotypen “<<interface>>”. Implementiert eine Klasse A die Schnittstelle B, so wird ein gestrichelter Pfeil mit ausgefüllter weißer Spitze von A nach B gezeichnet.



Interface comparable

Eines der meist genutzten Schnittstellen in Java ist **comparable**, welche die abstrakte Methode compareTo bereitstellt.

```
interface Comparable <T> {
   int compareTo(T o);
}
```

Diese Methode soll einen Wert zurückgeben, welcher:

- < 0 ist, wenn das aktuelle Objekt von Typ T “kleiner” als o ist.
- = 0 ist, wenn beide identisch sind.
- > 0 ist, wenn das aktuelle Objekt von Typ T “größer” als o ist.

Damit wird eine Totalordnung auf Objekten vom Typ T definiert.

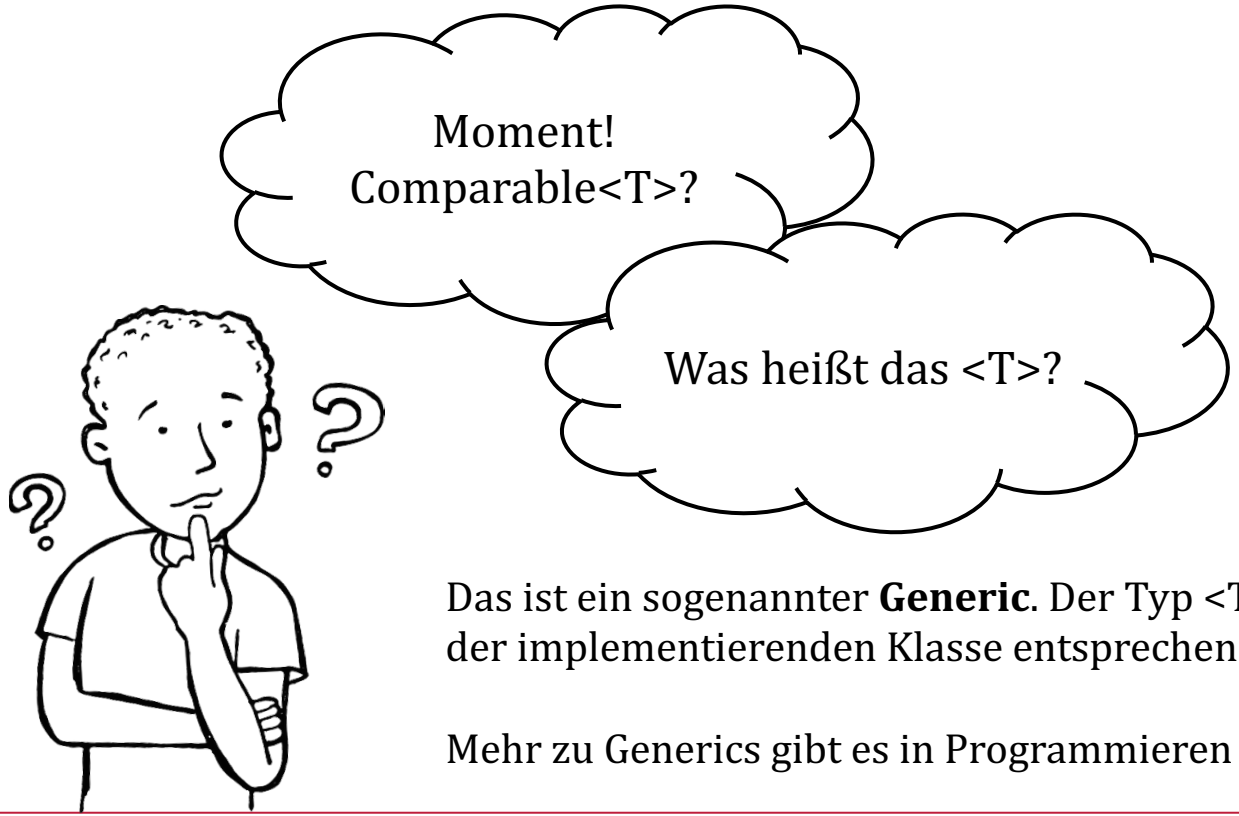
Comparable API-Auszug

> The natural ordering for a class C is said to be *consistent with equals* if and only if e1.compareTo(e2) == 0 has the same boolean value as e1.equals(e2) for every e1 and e2 of class C. […]
>
> It is strongly recommended (though not required) that natural orderings be consistent with equals.

Beispiel: Wie folgt könnte es sein, muss aber nicht.
Zwei Dreiecke x und y sind gleich (d.h. x.equals(y) == true), falls sie aus den drei selben Punkten bestehen.

x und y sind gleichgroß (d.h. x.compareTo(y) == 0), falls sie den gleichen Flächeninhalt besitzen.

Comparable<T>



Das ist ein sogenannter **Generic**. Der Typ <T> muss der implementierenden Klasse entsprechen.

Mehr zu Generics gibt es in Programmieren 2.

Comparable – Beispiel Strings

Die Klasse String implementiert Comparable und liefert darüber eine lexikographische Ordnung.

```
class String implements Comparable <String >, /* ... */ {
   /* ... */
}
```

```
String a = "abcdee";
String b = "abcdef";
System.out.println(a.compareTo(b)); // liefert negativen Wert
System.out.println(a.compareTo(a)); // liefert 0
System.out.println(b.compareTo(a)); // liefert positiven Wert
```

Die genauen Werte interessieren nicht, nur das Vorzeichen ist garantiert.

Comparable – Beispiel ??!!

```
public static Object getSmallest(Comparable[] objects) {
   Object smallest = objects [0];
   for (int i = 1; i < objects.length; ++i) {
      if (objects[i].compareTo(smallest) < 0) {
         smallest = objects[i];
      }
   }
   return smallest;
}
```

Diese Methode ist typunabhängig.

Es muss nur eine untereinander kompatible compareTo-Methode für die Objekte implementiert sein.

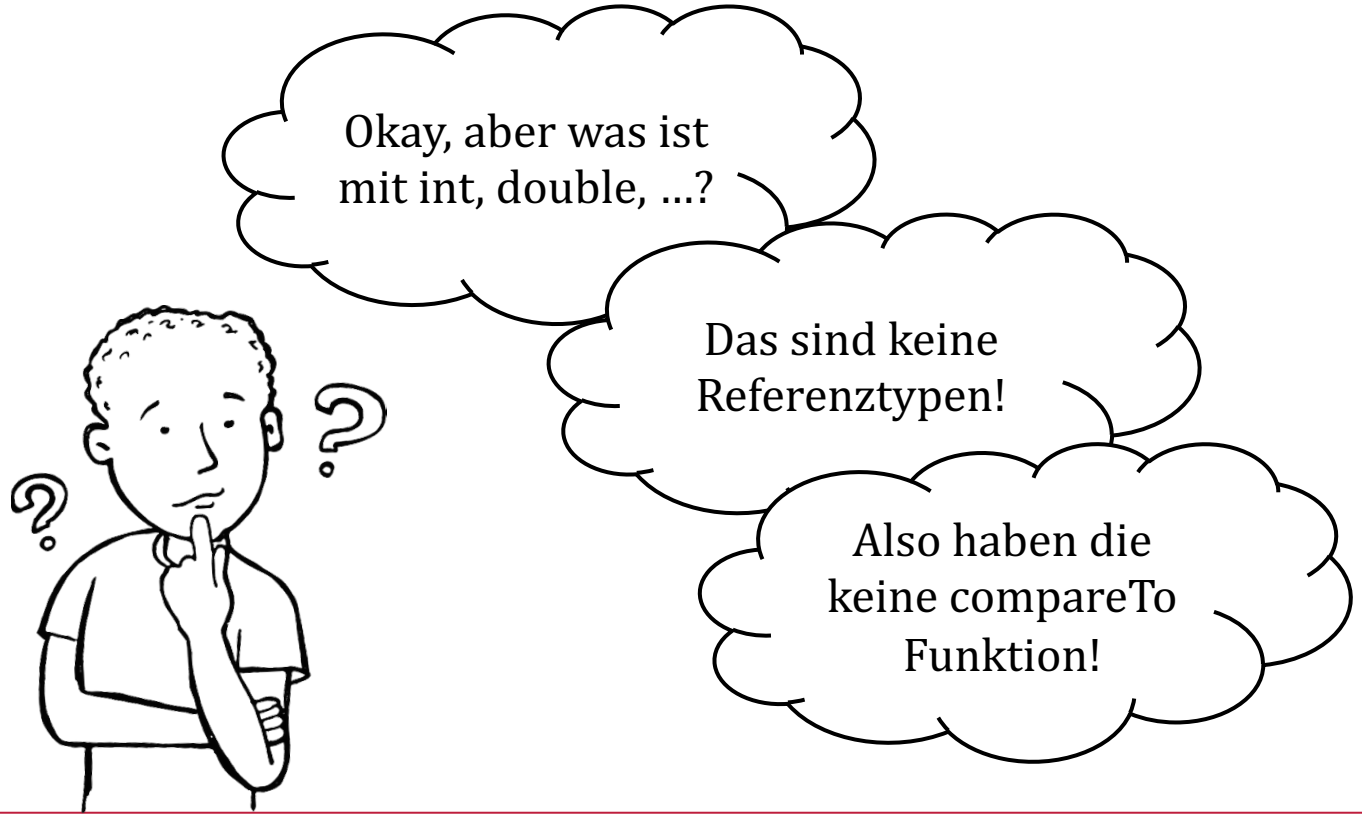

.compareTo(

)??

Dieser Code wirft schnell Probleme auf!

```
// Erzeugen und Ausgeben eines String -Arrays
Comparable[] objects = {
   "STRINGS", "SIND", "PAARWEISE", "VERGLEICHBAR"
};
for (int i = 0; i < objects.length; i++) {
   System.out.println((String) objects[i]);
}
// Ausgeben des kleinsten Elements
System.out.println((String) getSmallest(objects));
```

Primitive Datentypen



Wrapper Klassen

Eine **Wrapper-Klasse** kapselt einen primitiven Wert in einer objektorientierten Hülle und stellt Methoden zum Zugriff auf diesen zur Verfügung. Der Wert kann nicht verändert werden.

Damit können primitive Datentypen wie Referenztypen genutzt werden.

Jeder primitive Datentyp besitzt eine Wrapperklasse:
Integer, Float, Double, Boolean, …

Wrapper kann wie gewohnt über Konstruktoren initialisiert werden:
Integer i = new Integer(5);
Float f = new Float(3.141);
…

Initialisierung auch über Strings möglich: Integer(String s)

Wrapper Klassen – Integer und Float

Einige Methoden der Klasse Integer:

https://docs.oracle.com/javase/8/docs/api/java/lang/Integer.html

- int intValue()
- int String toString()
- int compareTo(Integer anotherInteger)
- static int compareUnsigned(int x, int y)
- static int parseInt(String s)
- static int parseInt(String s, int radix)

Einige Konstanten der Klasse Float:

https://docs.oracle.com/javase/8/docs/api/java/lang/Float.html

- MIN_VALUE
- MAX_VALUE
- NaN
- POSITIVE_INFINITY

Autoboxing und -unboxing

Java erlaubt automatisches boxing (Erstellen der Wrapperklasse) und unboxing (Herausziehen des Wertes).

Bei Integer kann auch manuell über intValue() auf den Wert zugegriffen werden.

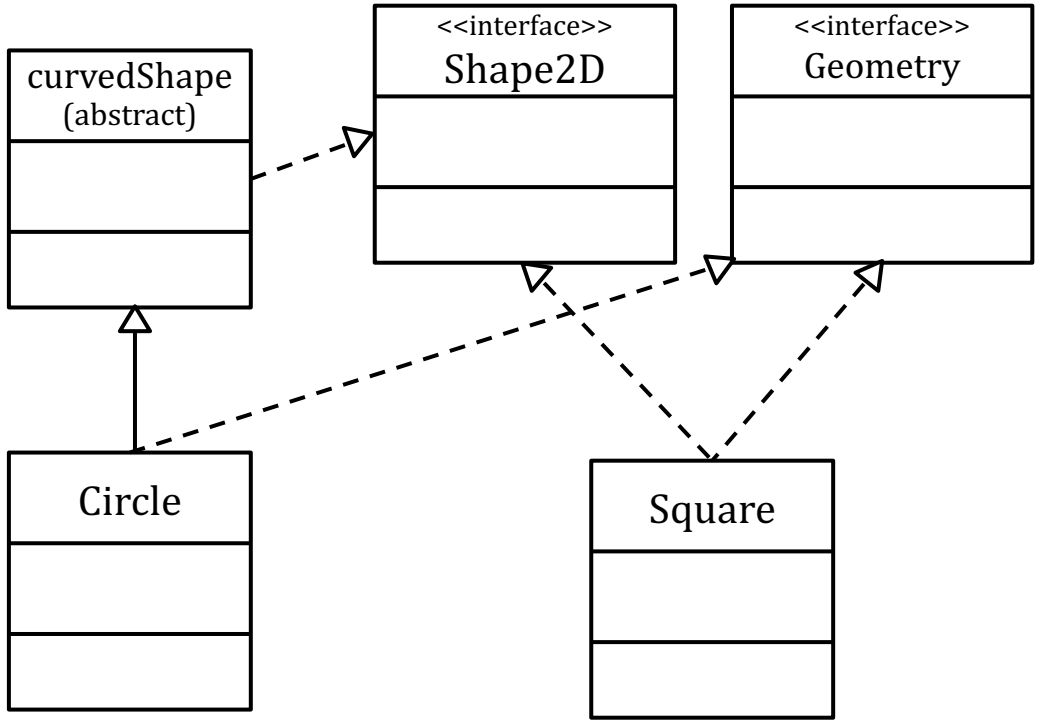
```
int n = 5;
Integer[] a = new Integer[n];
a[0] = 5;   // Autoboxing
a[1] = -5; // Autoboxing
a[2] = 0;  // Autoboxing
a[3] = 1;  // Autoboxing

int s = 0;
for (int i = 0; i < n; ++i) {
   s += a[i]; // Unboxing
}
```

Implementierung von Schnittstellen

Eine Klasse kann mehrere Schnittstellen implementieren.

Es muss nicht jede Methode implementiert sein. Die Klasse ist dann abstrakt.

Eine Klasse kann von einer Klasse erben und gleichzeitig eine Schnittstelle implementieren.



Schnittstellen – Beispiel für Fehler

```
interface A {
   int X = 1;
}
class B implements A {
   final static int X = 2;
   public static void main(String[] args) {
      System.out.println(X);
   }
}
class C extends B implements A {
   public static void main(String[] args) {
      System.out.println(X);
   }
}
```

Was wird bei dem Aufruf von C.main(null) ausgegeben?

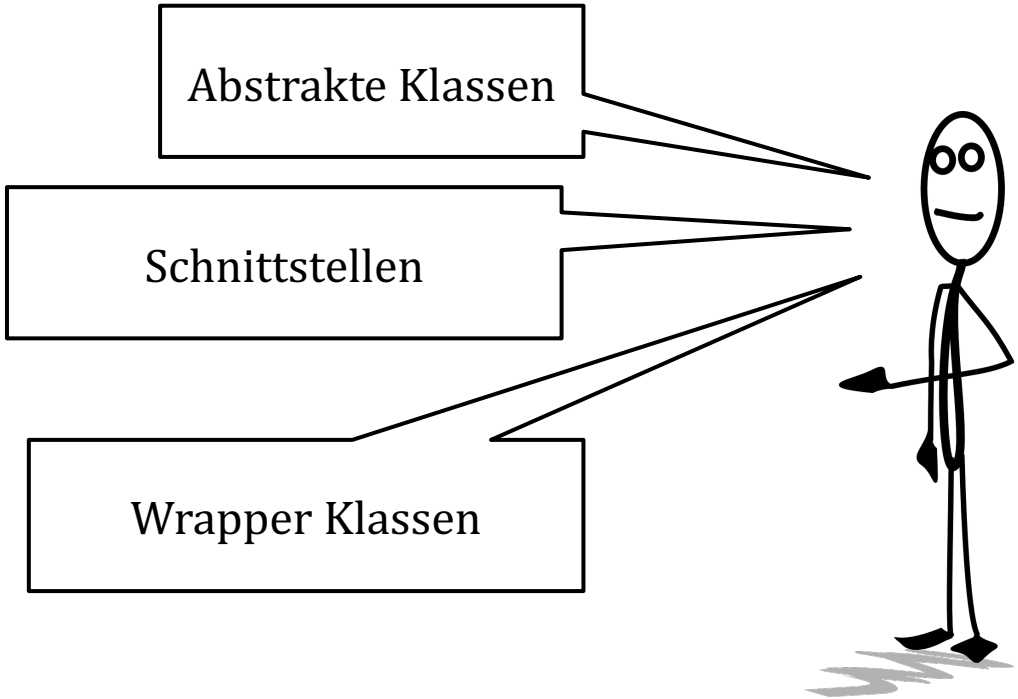
Fehler: reference to X is ambiguous

Konstanten in Schnittstellen

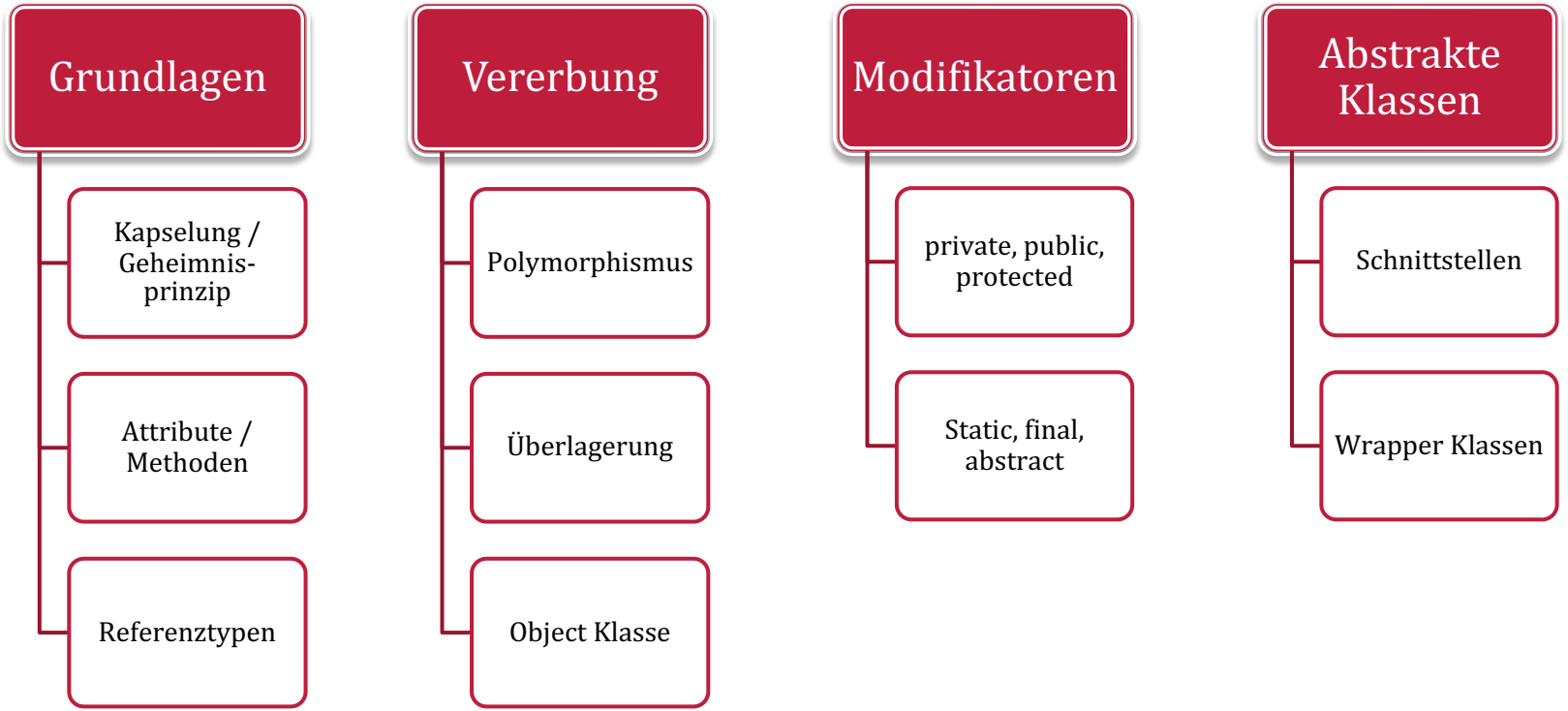
Konstanten können in implementierenden Klassen ohne Klassennamen direkt angesprochen werden. Damit können Konstanten gesammelt an einer Stelle definiert werden.

```
interface InterestingConstants {
    public static final float PI = 3.141f;
    public static final float E = 2.718f;
    public static final int C = 299_792_458;
}
```

Zusammenfassung



Objektorientierte Programmierung



Nächste Woche: Rekursion