

Software Engineering

6. Klassendiagramme

Franz-Josef Elmer, Universität Basel, HS 2013



Klassendiagramme

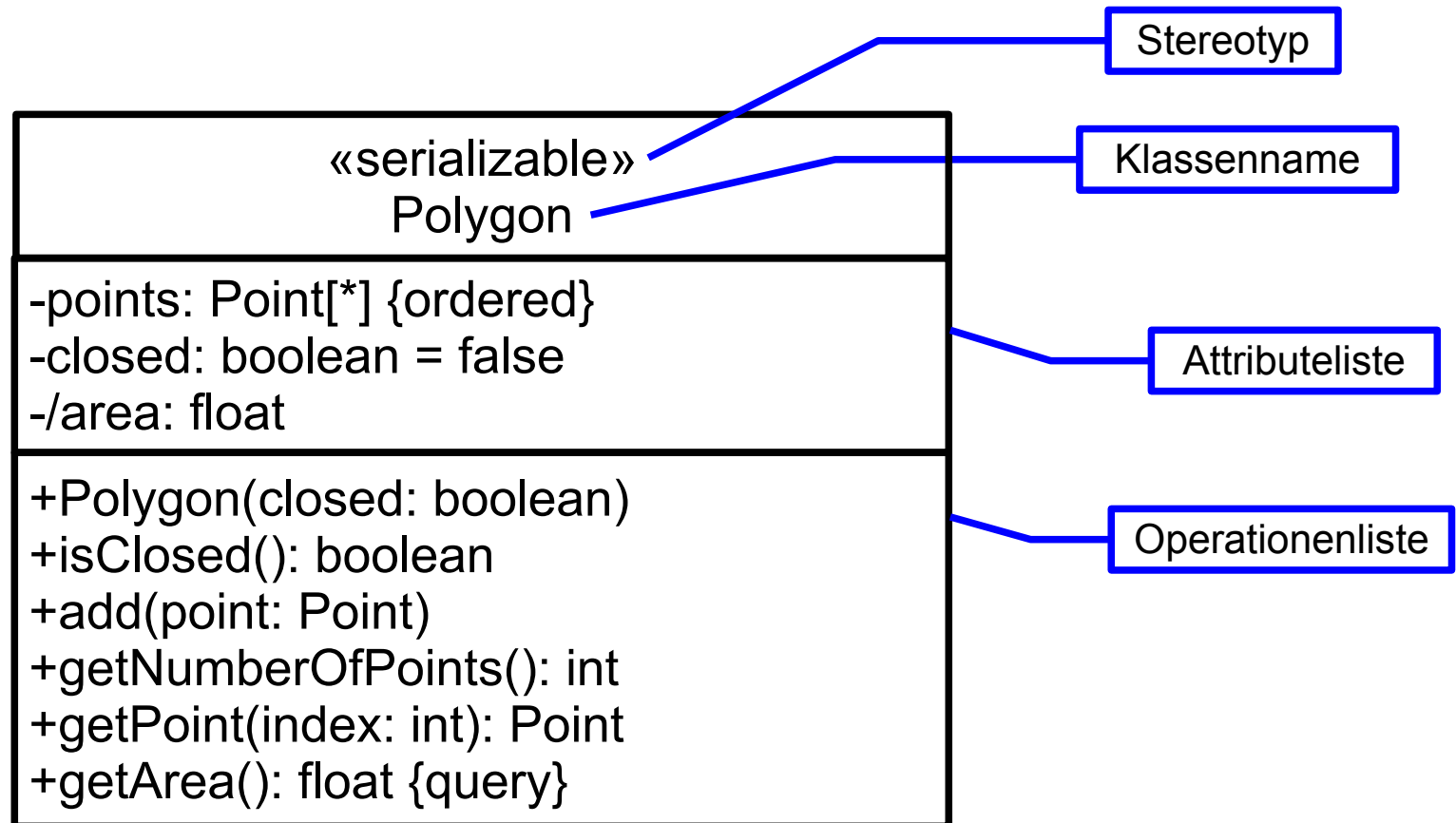
- Wichtigster Diagrammtyp in UML
- Statische objekt-orientierte Modelle des Systems
- Klassendiagramme zeigen
 - Attribute
 - Methodensignaturen (Operationen)
 - Vererbungshierarchien
 - Assoziationen und Abhängigkeiten zwischen Klassen
- Klassendiagramme müssen nicht komplett sein.
- Klassendiagramme sind ähnlich aber ausdrucksmächtiger als Entity-Relationship (ER) Diagramme aus der Daten(bank)welt.
- Automatisch generierte Klassendiagramme aus bestehender Software (*reverse engineering*) verschaffen einen Überblick über unbekanntem Code.

Das Klassensymbol

A diagram of a UML class symbol, which is a rectangle with a black border. The word "Klasse" is centered inside the rectangle in a large, black, sans-serif font.

Klasse

Das Klassensymbol vollgepackt



Namensfeld



```
classDiagram
    class Polygon {
        <<serializable>>
    }
    style Polygon fill:#fff,stroke:#000,stroke-width:2px
```

«serializable»
Polygon

- Klassenname:
 - zentriert
 - kursive gesetzter Name = abstrakte Klasse
- Beispiel:

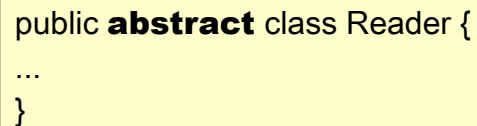
UML



```
classDiagram
    class Reader {
    }
    style Reader fill:#fff,stroke:#000,stroke-width:2px
```

Reader

Java



```
public abstract class Reader {
    ...
}
```

Stereotyp



«serializable»
Polygon

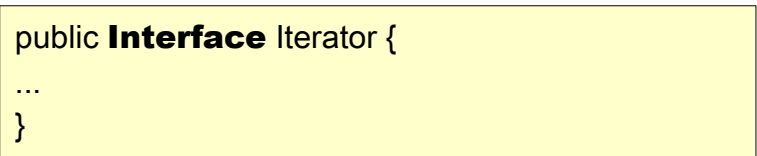
- Stereotyp (in UML 2 'Schlüsselwort'):
 - Erweiterungsmechanismus in UML zur Definition neuer Sprachelemente.
 - Stereotyp (in franz. Anführungszeichen) wird dem Element hinzugefügt.
 - Vordefinierte Stereotypen wie z.B. «Interface»
- Beispiel:

UML



«Interface»
Iterator

Java



```
public Interface Iterator {  
    ...  
}
```

Stereotyp: Beispiel «serializable»

«serializable»
Polygon

- Beispiele der möglichen Bedeutung von «serializable»:

Java Serialisierung:

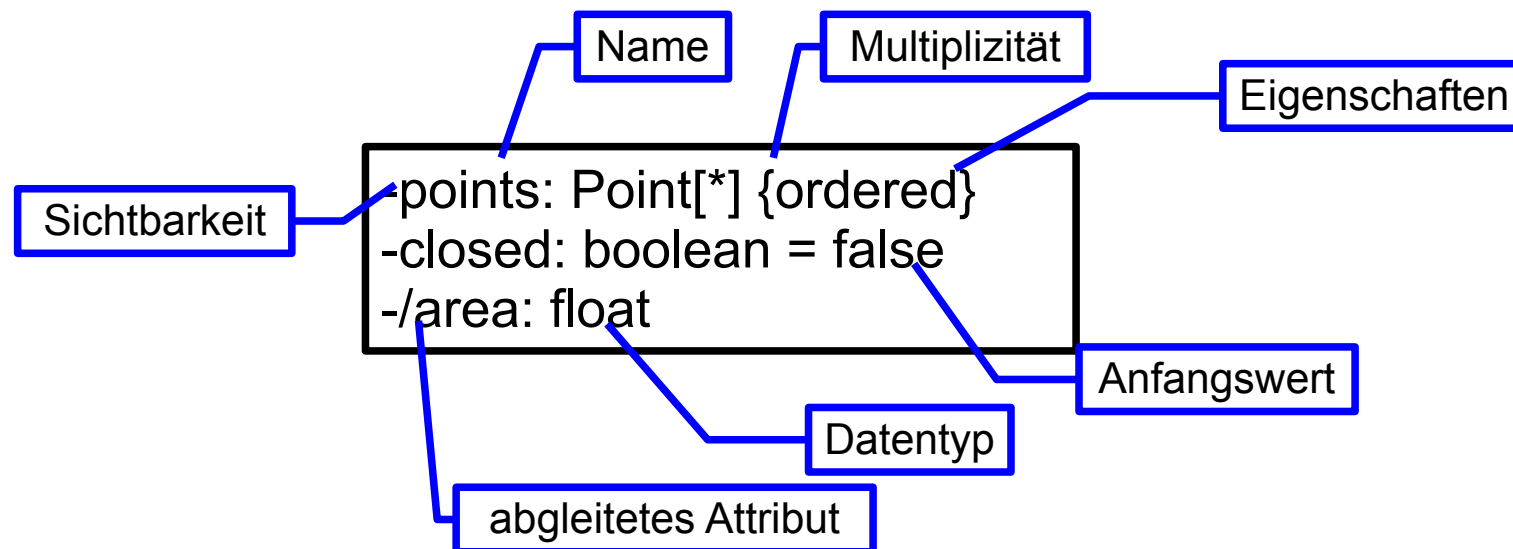
```
public class Polygon implements java.io.Serializable {  
    ...  
}
```

XML Serialisierung:

```
import org.w3c.dom.Element;  
  
public class Polygon {  
    public Polygon(Element serializedPolygon) {  
        ...  
    }  
  
    public Element serialize() {  
        ...  
    }  
    ...  
}
```

Attribute

- Beschreibung der Daten und Zustände, die eine konkrete Instanz der Klasse annehmen kann.
- Attribute sind linksbündig gesetzt.
- Eine Attributedefinition pro Zeile.
- Es müssen nicht alle Attribute aufgeführt sein.
- Bestandteile:



Attributsyntax

[visibility] ['/'] name [':' type] ['[' multiplicity ']'] ['=' default] ['{' att-props '}']

visibility ::= '+' | '#' | '~' | '-'

multiplicity ::= [lower '..'] upper

lower ::= integer

upper ::= '' | integer*

*att-props ::= att-property [',' att-property]**

*att-property ::= 'readOnly' | 'frozen' | 'union' | 'subsets' attribute-name
| 'redefines' attribute-name | 'ordered' | 'unique' | constraint*

Attributsemantik

- Name (*name*):
 - Eindeutiger Attributbezeichner.
 - Entspricht meistens dem Feldnamen in einer Programmiersprache.
- Sichtbarkeit (*visibility*):
 - + = public, # = protected, ~ = package protected, - = private
 - Achtung: Die Sichtbarkeitssemantik der UML Spezifikation ist nicht unbedingt dieselbe einer Programmiersprache.
 - Fehlendes Sichtbarkeitssymbol bedeutet Sichtbarkeit ist
 - unbekannt (z.B. in einer frühen Designphase) oder
 - irrelevant.
- Abgeleitetes Attribut ('/'):
 - Attribut dessen Wert aus anderen Attributen ableitbar ist.
 - Beispiel: Fläche eines geschlossenen Polygons ist eindeutig durch die Lage seiner Eckpunkte bestimmt.
- Typ (*type*):
 - Datentyp des Attributs.
 - Entspricht meistens dem Datentyp in einer Programmiersprache.

Attributsemantik

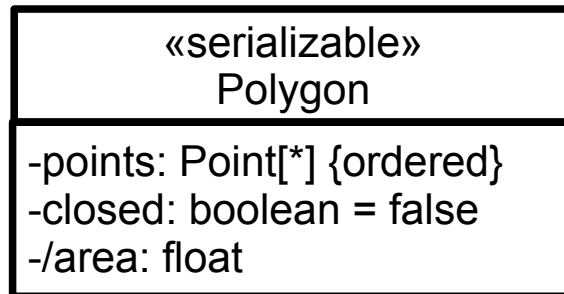
- Multiplizität (*multiplicity*):
 - Zeigt an wieviele Objekte zum Attribut gehören können.
 - Allgemeine Form: $n..m$
mit $n \geq 0$, $m \geq 1$ und $m \geq n$
D.h. das Attribut hat mindestens n maximal m Objekte des angegebenen Typs.
 - Sonderformen:
 - $n \triangleq n..n$: Exakt n Objekte
 - $n..*$: Beliebig viele Objekte (Minimum n)
 - $* \triangleq 0..*$: Keine Einschränkungen über die Zahl der Objekte
 - keine Angabe $\triangleq 0..1$ oder 1
 - Übersetzung von Begriffen, die sich auf Attributmultiplizität beziehen (Diese findet man z.B. in Requirements):
 - Optional: $0..n$
 - Obligatorisch (engl. *mandatory*): $n \geq 1$
 - Einwertig: $0..1$ oder 1
 - Mehrwertig: $*$, $n..*$ oder $n..m$ mit $m \geq 2$

Attributsemantik

- Anfangswert (*default*):
 - Wert des Attributs nach der Erzeugung
 - Optional
 - Bei Attribute des Typs String muss der Anfangswert zwischen zwei doppelten Anführungszeichen "" stehen.
Beispiel: - title: String = "untitled"
- Eigenschaft (*att-property*):
 - Zusätzliche Eigenschaft eines Attributs:
 - readOnly: Attribut kann von aussen nur gelesen werden.
 - frozen: Attributwert kann nur einmal gesetzt werden.
 - ordered: Geordnete Objekte (bei mehrwertigen Attributen)
 - unique: Enthält keine Duplikate (bei mehrwertigen Attributen)
 - Liste aus Eigenschaften ist optional.
 - Eigenschaft als Einschränkung (*constraint*) an den Wertebereich eines Attributs: Eine Bedingung, die für alle erlaubten Werte wahr (true) sein muss.
Beispiel: - weight: float {weight >= 0}

Attribute umsetzen in Java

- Oft verschiedene Möglichkeiten der Umsetzung.
- Beispiel:



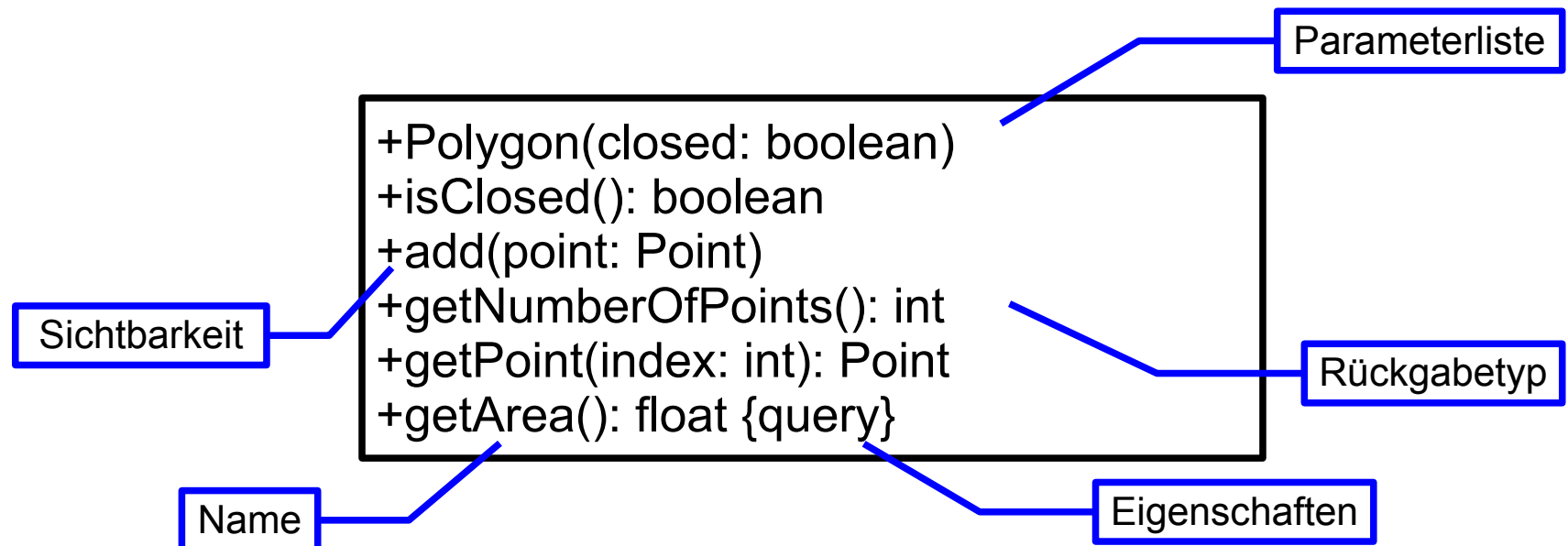
```
public class Polygon implements java.io.Serializable {  
    private Point[] points;  
    private boolean closed;  
    private transient float area;  
    ...  
}
```

```
public class Polygon implements java.io.Serializable {  
    private List<Point> points = new ArrayList<Point>();  
    private boolean closed;  
    ...  
}
```

- Empfehlungen für mehrwertige Attribute:
 - {ordered}: Typ[] oder List<Typ> (ArrayList oder LinkedList)
 - {unique}: Set<Typ>
 - {ordered, unique}: SortedSet<Typ> oder LinkedHashSet<Typ>

Operationen

- Eine Operation ist eine ausführbare Aktion:
 - Operation \neq Methode
 - Operation \sim Methodendeklaration
- Operationen sind linksbündig gesetzt.
- Abstrakte Operationen sind kursive gesetzt.
- Eine Operationsdefinition pro Zeile.
- Es müssen nicht alle Operationen aufgeführt sein.
- Bestandteile:



Operationssyntax

[visibility] name '(' [parameter-list] ')' [':' [return-type] [{' op-properties '}']]

*parameter-list ::= parameter [',' parameter]**

parameter ::= [direction] parameter-name ':' type

['['multiplicity''] ['=' default]

[{' parm-property [',' parm-property] '}']*

direction ::= 'in' | 'out' | 'inout'

*op-properties ::= op-property [',' op-property]**

op-property ::= 'query' | 'ordered' | 'unique' | 'redefines' op-name | constraint

Operationssemantik

- Name (*name*):
 - Bezeichner der Operation.
 - In der frühen Entwurfsphase auch nur eine kurze Beschreibung (ohne Parameterklammern).
 - Entspricht meistens dem Funktions- bzw. Methodennamen in einer Programmiersprache.
- Sichtbarkeit (*visibility*): Wie für Attribute
- Parameterliste (*parameter-list*):
 - Kommaseparierte Liste der Parameter (*parameter*).
 - Parametersyntax und -semantik analog zu Attributen
 - Optionaler Richtung (*direction*) vor dem Namen:
 - in: Inputparameter (Standard wenn keine Richtung angegeben)
 - out: Outputparameter
 - inout: Sowohl Input- wie Outputparameter
- Rückgabetyt (*return-type*):
 - Datentyp des Rückgabewertes (falls es einen gibt)
 - Statt Typ[*] auch Typ[] (Array), List etc.

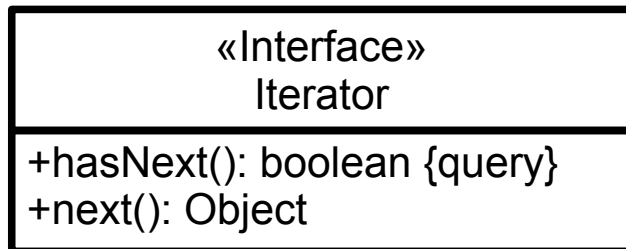
Operationensemantik

- Eigenschaft (*op-property*):
 - Zusätzliche Eigenschaft einer Operation oder des Rückgabewertes:
 - query: Operation ändert den sichtbaren Zustand nicht.
 - ordered: Geordneter Rückgabewert
 - unique: Rückgabewert ohne Duplikate
 - Eigenschaftsliste ist optional.
 - Eigenschaft als Einschränkung (*constraint*) an die Operation.

Operationen umsetzen in Java

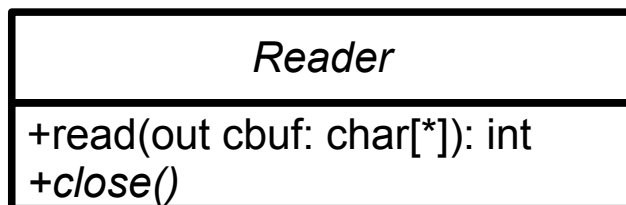
- Da Operation \neq Methode kann nur ein Methodenrumpf bzw. Methodensignatur umgesetzt werden.
- Beispiele:

UML



Java

```
public interface Iterator {  
    /**  
     * Returns <code>>true</code> if not finished.  
     * This method should not change the state of  
     * this iterator.  
     */  
    public boolean hasNext();  
    public Object next();  
}
```

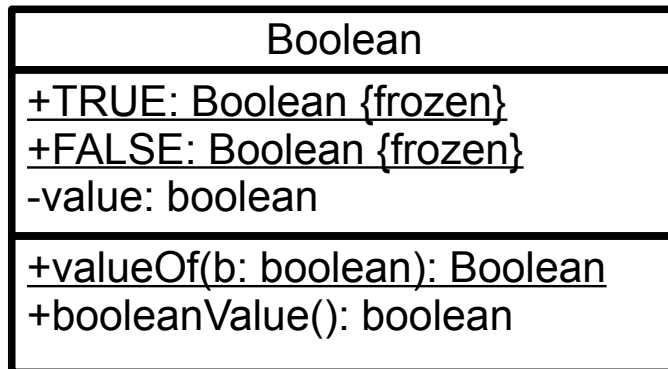


```
public abstract class Reader {  
    public int read(char[] cbuf) {  
        ...  
    }  
    public abstract void close();  
    ...  
}
```

Klassenattribute und -operationen

- Attribute und Operationen, die sich nicht auf eine Instanz (Objekt) der Klasse beziehen sondern auf die Klasse selber (*static attributes* und *static operations*), werden unterstrichen dargestellt.
- Beispiel:

UML

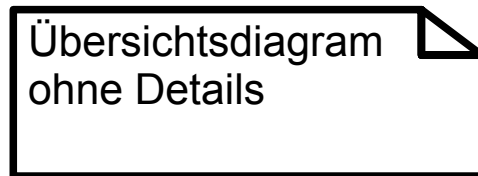


Java

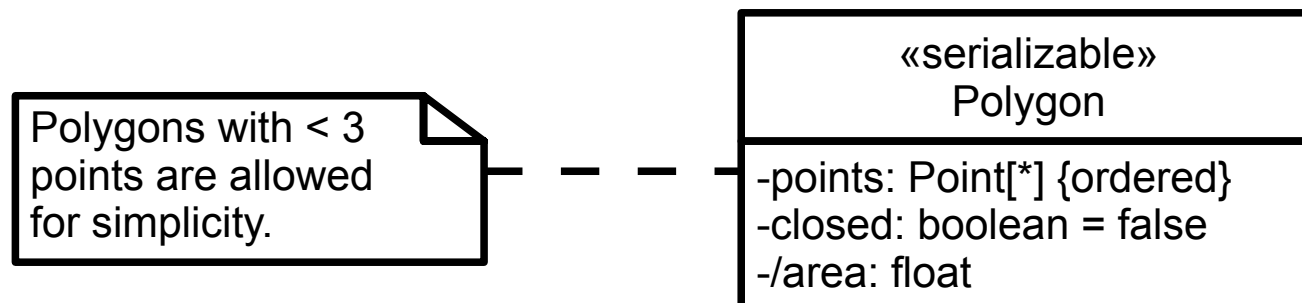
```
public class Boolean {  
    public static final Boolean TRUE  
        = new Boolean(true);  
    public static final Boolean FALSE  
        = new Boolean(false);  
    private boolean value;  
  
    public static Boolean valueOf(boolean b) {  
        return (b ? TRUE : FALSE);  
    }  
  
    public boolean booleanValue() {  
        return value;  
    }  
}
```

Notiz

- Notizen sind Kommentare in allen UML Diagrammen
- Notation:

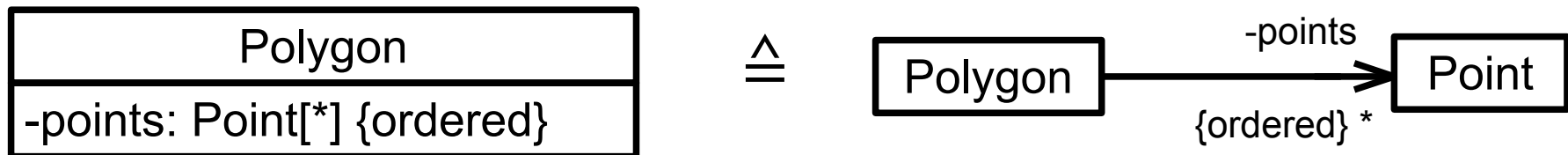


- Eine Notiz, welche durch eine gestrichelten Linie mit einem Diagrammelement verbunden ist, kommentiert nur dieses.



Assoziationen

- Alternative Darstellung von Attributen, welche die Verbindung von Objekten verschiedener Klassen betont.
- Beispiel:



- Assoziation verbindet zwei Klassensymbole.
- Beide Enden haben optional
 - Sichtbarkeit und Name
 - Eigenschaften und Multiplizität
 - Pfeilspitze
- Pfeile zeigen Navigierbarkeit an:
 - Instanzen der Klasse an der Pfeilspitze sind Attribute der Klasse am anderen Ende.
- Pfeillose Assoziation bedeutet:
 - Doppelpfeil oder
 - Navigierbarkeit ist unbekannt oder irrelevant

Assoziationen: Beispiele



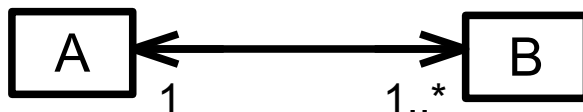
- Bidirektionale Assoziation zwischen A und B.
- A ist Attribut von B und umgekehrt.
- Paare von A und B Objekten referenzieren sich gegenseitig.



- B ist ein obligatorisches Attribut von A
- B hat kein Attribut vom Typ A.
- Jede Instanz von B wird von genau einer Instanz von A referenziert.

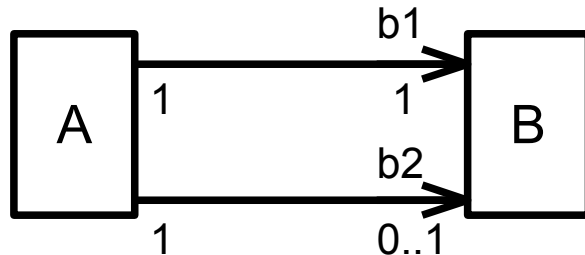


- A hat ein einwertiges Attribute vom Typ B welches leer sein kann.
- B hat kein Attribut vom Typ A.
- Instanzen von B können von beliebig vielen Instanzen von A referenziert werden.

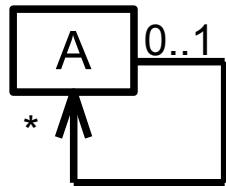


- A Objekte referenzieren mindestens ein B Objekt.
- Jedes B Objekt referenziert sein A Objekt in dem es enthalten ist.
- B Objekte werden von nur einem A Objekt referenziert.

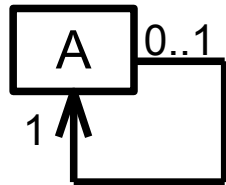
Assoziationen: Beispiele



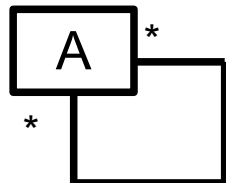
- A hat zwei Attribute vom Typ B (b1 und b2).
- b1 ist obligatorisch und b2 optional.
- Jede Instanz von B wird von genau einer Instanz von A referenziert.



- A hat ein mehrwertiges Attribute vom selben Typ.
- Jede Instanz A wird nur von höchstens einer Instanz von A referenziert.
- Baumstruktur



- Verkettete Liste

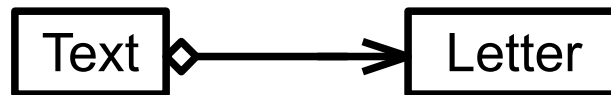


- Graph

Aggregation und Komposition

- Aggregation:  oder 

- Ist-Teil-von Beziehung
- Beispiel:



- Komposition:  oder 

- Exklusive ist-Teil-von Beziehung: Wenn das Ganze gelöscht wird, werden auch seine Teile gelöscht.
- Beispiel:



- Aggregation versus Komposition:

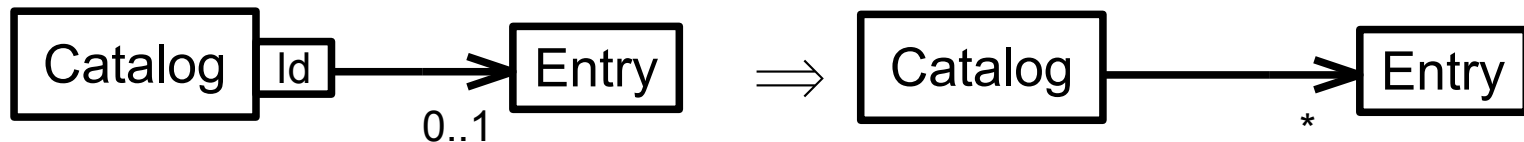
- Aggregation: Teile können von mehreren Ganzen referenziert werden:



- Komposition impliziert  oder 

Qualifizierte Assoziation

- Mehrwertiges Attribut als Menge von Schlüssel-Werte Paare
- Beispiel:



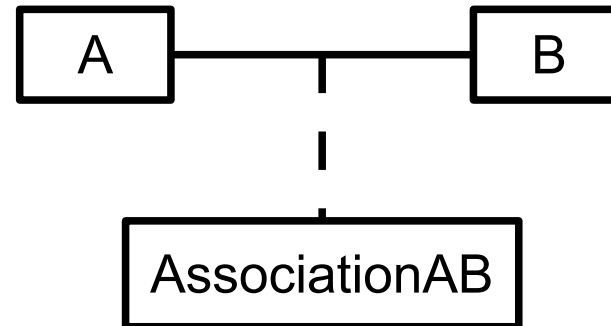
- Der Schlüssel vom Typ **Id** liefert eine oder keine Instanz von **Entry**.
- Umsetzung in Java:
 - Klassen, die das Interface **Map** implementieren.
 - Beispiel:

```
public class Catalog {  
    private Map entries = new HashMap();  
    public Entry getEntry(Id id) {  
        return (Entry) entries.get(id);  
    }  
    ...  
}
```

Assoziationsklasse

- Assoziation als Klasse mit Attributen und Operationen:

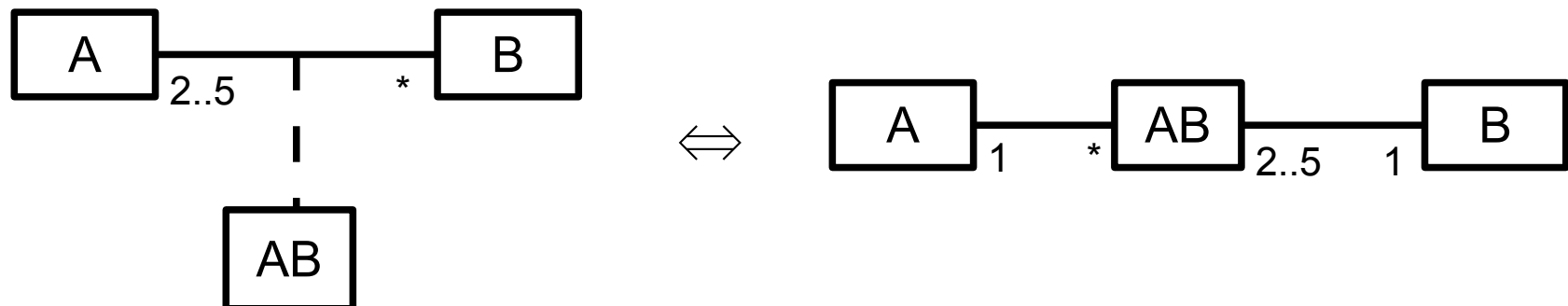
- Notation:



- Semantik:

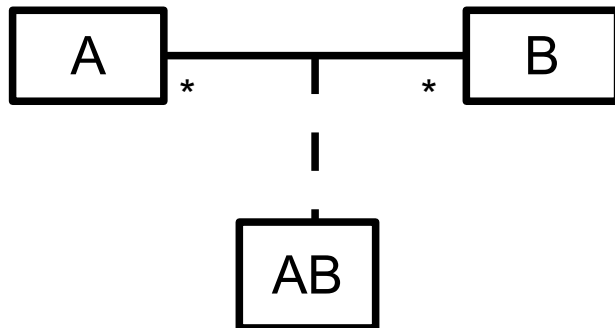
- Zu jedem Paar von Instanzen von A und B gehört eine Instanz der Assoziationsklasse AssociationAB.

- Alternative Notation:



Assoziationsklasse

- Mögliche Umsetzung in Java:



```

public class A { // similar for B
    private List<AB> associations = new ArrayList<AB>();
    public void addAB(AB ab) {
        associations.add(ab);
    }
    ...
}

```

```

public class AB {
    public static void connect(A a, B b) {
        AB ab = new AB(a, b);
        a.addAB(ab);
        b.addAB(ab);
    }
    private final A a;
    private final B b;
    ...
    private AB(A a, B b) {
        this.a = a;
        this.b = b;
    }
    ...
}

```

// client code

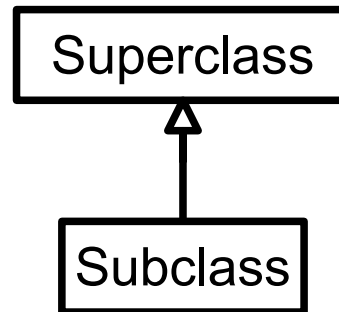
```

...
A a = new A();
B b = new B();
AB.connect(a, b);
...

```

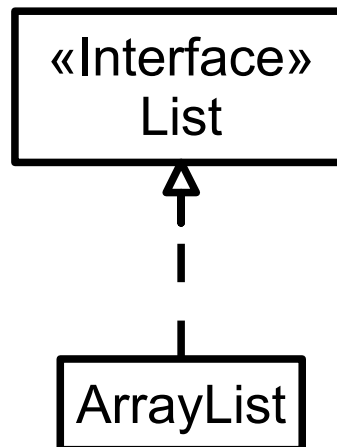
Vererbung und Implementierung

- Vererbung:



- In Java: `class Subclass extends Superclass`

- Implementierung:



- In Java: `public class ArrayList implements List`

Abhängigkeiten zwischen Klassen

Vier verschiedene Pfeile, die zeigen das A von B abhängt:

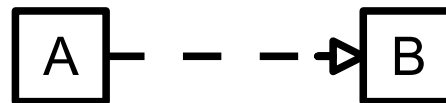
– Assoziation:



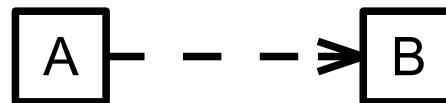
– Vererbung:



– Implementierung:



– andere Abhängigkeit:



- Stereotypen zur näheren Bezeichnung der Art der Abhängigkeit:
 - «call»: A ruft Operation von B.
 - «create»: A erzeugt Instanz von B.
 - «parameter»: Eine Operation von A hat ein Parameter vom Typ B.

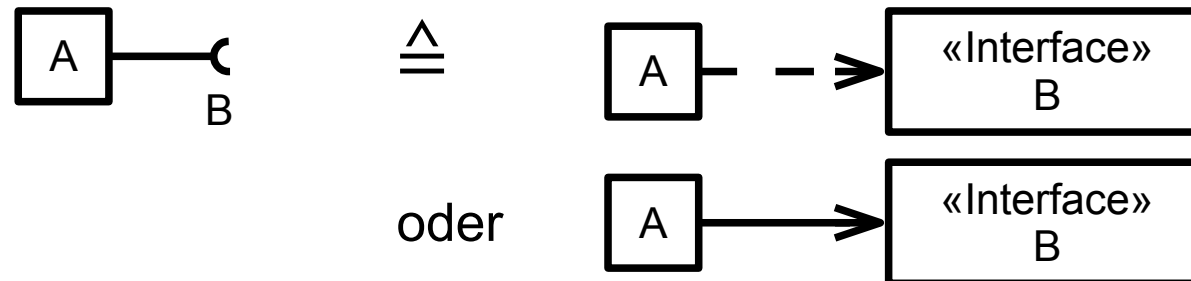
Abhängigkeiten von Schnittstellen

Spezielle Notationen:

- Klasse A bietet Schnittstelle B an:

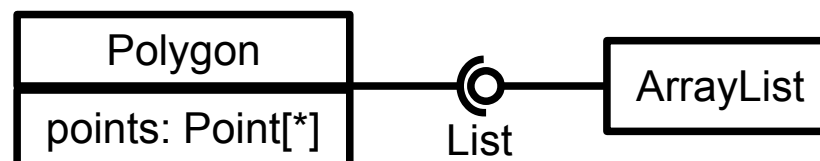


- Klasse A benötigt Schnittstelle B (neu in UML2):



- Kombiniert:

- Beispiel:



Parametrisierte Klassen

- Klassen mit Platzhalter für Typen:
 - Platzhalter in gestricheltem Kasten oben rechts.
 - Konkrete Klasse „implementiert“ die parametrisierte Klasse.
 - Stereotyp «bind» zusammen mit der Platzhaltersubstitution legt die konkrete Klasse fest.
 - Der konkreten Klasse dürfen keine Attribute oder Operationen hinzugefügt werden bzw. überschrieben werden.
- C++: Templates
- Java 1.5: Generics

