

Objektorientierte Software-Entwicklung

Priv.-Doz. Dr. Rolf Hennicker

08.01.2003

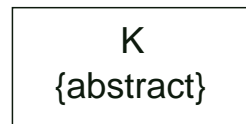
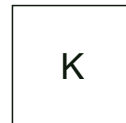


Implementierung von Klassendiagrammen in Java

Implementierung von Klassendiagrammen in Java

1. Klassen und Schnittstellen deklarieren

UML

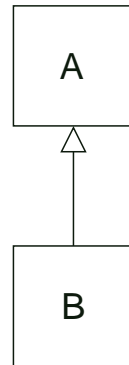


Java

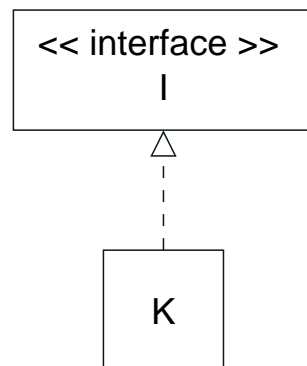
```
class K {...}
```

```
abstract class K {...}
```

```
interface I {...}
```

UML*Java*

```
class B extends A {...}
```



```
class K implements I {...}
```

2. Attribute deklarieren

<i>UML</i>	<i>Java</i>
attribut:Typ	JavaTyp attribut;

wobei UML-Basistypen folgendermaßen in Java-Typen übersetzt werden:

<i>UML</i>	<i>Java</i>
Boolean	boolean
Integer	int
Real	float oder double
String	String

3. Methodenköpfe deklarieren

UML

op(x: Typ)

op(x: Typ): ResTyp

op() {abstract}

K(x: Typ)

Java

```
void op(JavaTyp x) {...};
```

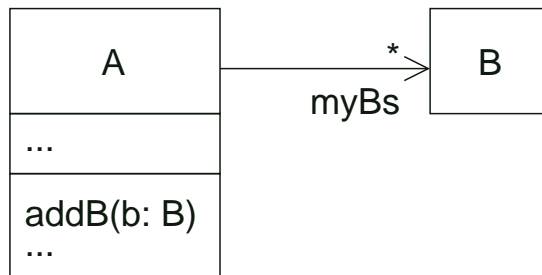
```
JavaResTyp op(JavaTyp x) {...};
```

```
abstract void op() {...};
```

```
K(JavaTyp x) {...}; //Konstruktor
```

4. Assoziationen darstellen

UML



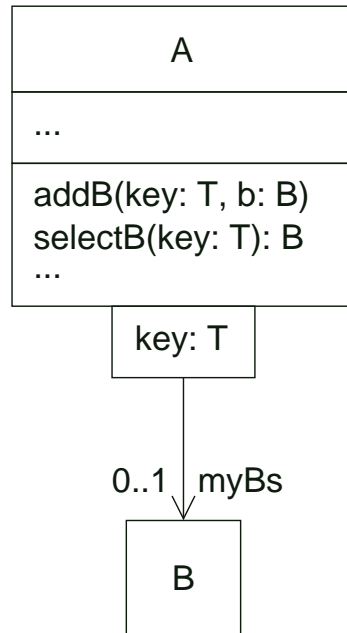
Java

```
class A {
    B myB; //Referenzattribut
    ...
}
```

```
// Vector = dynamische Felder, die
// beliebige Objekte enthalten können
```

```
import java.util.Vector;

class A {
    Vector myBs = new Vector();
    ...
    void addB(B b) {
        myBs.add(b);
    }
    ...
}
```

UML*Java*

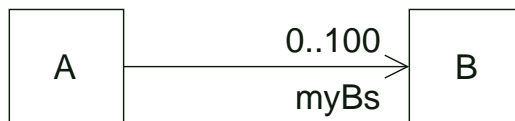
```

//Hashtable speichert Schlüssel/Element-Paare
import java.util.Hashtable;

class A {
    Hashtable myBs = new Hashtable();
    ...
    void addB(T key, B b) {
        myBs.put(key, b);
    }
    B selectB(T key) {
        return (B) myBs.get(key);
    }
    ...
}

```

Beachte: Falls $T = \text{int}$ ist, muss als aktueller Parameter statt `key` `new Integer(key)` verwendet werden.



```

class A {
    B[] myBs = new B[100];
    ...
}

```

5. Zugriffsrechte bestimmen

UML

Java

-

private

#

protected

//in Subklassen und im selben Paket sichtbar

+

public