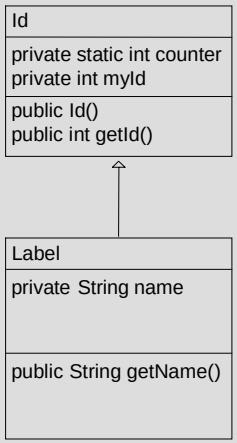
Informatik I – Kapitel 8

„ Höhere objektorientierte Konzepte“

Zusammenfassung des Kapitel 8

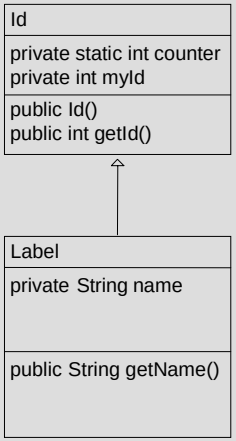
Küchlin, Weber, Einführung in die Informatik, 2.Auflage

25.1.2004



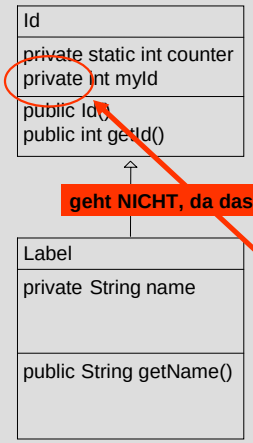

```
public class Id {
   private static int counter = 1;
   private int myId;
   public Id(){ myId = counter++; }
   public int getId(){ return myId; }
}
```

```
public class Label extends Id {
  private String name;
  public Label() {
    // super(); // -> Id(); implizit!
    name = "unknown";
  }
  public Label(String val) {
    name = val;
  }
  public String getName(){
    return name;
  }
}
```

Die Methoden der Klasse Id, von der geerbt wird, sind in Label auch automatisch vorhanden!

```
public class Main {
  public static void main(String[] args) {
    Label myLabel = new Label("as2004");

    System.out.println(
    “Das Gerät“ + myLabel.getName() +
    “hat die Id“+ myLabel.getId());
  }
}
```


```
public class Id {
  ...
  private int myId;
  ...
}
```

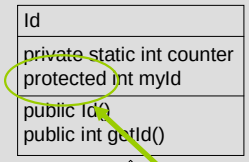
geht NICHT, da das Feld private ist!

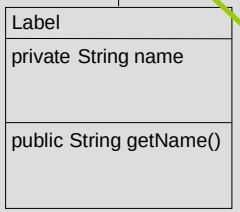
```
public class Label extends Id {
  ...
  public Label(String val) {

    name = val;

    super.myId = 0; //hier: ⇔ myId = 0;

  }
  ...
}
```

Vererbung/ Sichtbarkeit

5

marc-oliver pahl





geht, da das Feld protected ist!

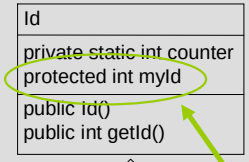
```
public class Id {
  ...
  private int myId;
  ...
}
```

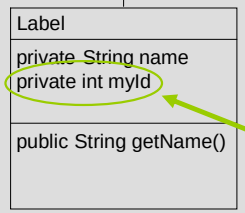
```
public class Label extends Id {
  ...
  public Label(String val) {
    super();
    name = val;

    super.myId = 0; //hier: ⇔ myId = 0;

  }
  ...
}
```

Vererbung/ Sichtbarkeit

6

marc-oliver pahl






```
public class Id {
  ...
  private int myId;
  ...
}
```

```
public class Label extends Id {
  ...
  public Label(String val) {
    super();
    name = val;

    super.myId = 0; // NICHT = myId = 0;

    myId = 23;
  }
  ...
}
```

virtuelle Funktionen/ Überschreiben

7

marc-oliver pahl

```
public class Parent {
  public void method(){
    System.out.println("Methode des Parent.");
  }
}

public class Child extends Parent {
  public void method(){
    System.out.println("Methode des Child.");
  }
}


public class Main {
  public static void main(String[] args) {
    Parent eins = new Parent();
    Parent zwei = new Child();
    eins.method();

    zwei.method();

  }
}
```

virtuelle Funktionen/ Überschreiben/ final

8

marc-oliver pahl

```
public class Parent {
  public final void method(){
    System.out.println("Methode des Parent.");
  }
}

public class Child extends Parent {
  public void method(){
    System.out.println("Methode des Child.");
  }
}


public class Main {
  public static void main(String[] args) {
    Parent eins = new Parent();
    Parent zwei = new Child();
    eins.method();

    zwei.method();

  }
}
```

Kundenklasse

```
public class Main {
  public static void main(String[] args){
    ZahlenSpeicher z1 = new ZahlenSpeicher1();
    ZahlenSpeicher z2 = new ZahlenSpeicher2();
    z1.setValue(7); z2.setValue(8);
    System.out.println("z1+z2="+(z1.getValue()+z2.getValue()));
  }
```

Schnittstelle/ Interface

```
public interface ZahlenSpeicher {
public int getValue();
public void setValue(int val);}
```

interne Repräsentation/ Implementierung

```
public class ZahlenSpeicher1 implements ZahlenSpeicher {
  private int theValue = 0;
  public int getValue(){ return theValue; }
  public void setValue(int val){ theValue = val; }
}
```

```
public class ZahlenSpeicher2 implements ZahlenSpeicher {
  private String theValue;
  public int getValue(){ return Integer.parseInt(theValue); }
  public void setValue(int val){ theValue = Integer.toString(val); }
}
```


- Stichwort: Muster
  - in C++

```
template<class T> class Node{
  private:
    T data;
    Node* next;
  public:
    T getData();
  ...
}
```

  - in Java

```
public static void inc(ZahlenSpeicher z){
    z.setValue(z.getValue()+1);
}
```

Funktioniert „generisch“ für alle beliebigen Implementierungen des Interface ZahlenSpeicher