

Weiteres zur OO

((itanius informatik))

Java Foundation Track by Carsten Bokeloh

Weiteres zur OO

- Pakete
- Zugriffsrechte
- Innere Klassen
- Anonyme Klassen

Pakete

- ❖ stellt eine Sammlung von Klassen dar
 - ❖ `java.lang` oder `de.lhsystems.customer`
- ❖ `javac -d . mypackage/myclass.java`
- ❖ `import de.lhsystems.customer.*;`


Zugriffsrechte

public - zugreifbar in allen Klassen

protected - zugreifbar in allen Subklassen

package(kein Modifikator)
zugreifbar in Klassen
des Paketes

private
zugreifbar
in der Klasse

-
- 
-
- ❖ Möglichkeit, Klassen innerhalb von Klassen zu definieren
 - ❖ sehr oft verwendet in der Programmierung grafischer Oberflächen zur Behandlung von Mausklicks etc.
 - ❖ weitere Möglichkeit, um Details zu verstecken

Inner Klassen

Typ	Beispiel
statische innere Klasse	<code>class Out static class In {}</code>
Mitgliedsklasse	<code>class Out class In {}</code>
lokale Klasse	<code>class Out Out() class In {}</code>
anonyme innere Klasse	<code>class Out Out() new Runnable() public void run() {};</code>

Statische Innere Klassen

```
public class NestedClassExample2 {  
  
    public class InnerClass {  
        int b = 4;  
  
        public void method2() {  
            System.out.println("World!");  
        }  
    }  
  
    public void method1() {  
        InnerClass ic = new InnerClass();  
        System.out.println(ic.b);  
    }  
  
    public static void main(String[] args) {  
        NestedClassExample2 nce2 = new NestedClassExample2();  
        nce2.method1();  
  
        NestedClassExample2.InnerClass ic2 = nce2. new InnerClass();  
        System.out.println(ic2.b);  
        ic2.method2();  
    }  
}
```

Statische Innere Klassen

- ❖ wird oft verwendet in `java.util`
- ❖ Collection Klassen, die wir später behandeln

Nicht statische Member Klassen

```
public class NestedClassExample2 {
```

```
    public class InnerClass {  
        int b = 4;  
  
        public void method2() {  
            System.out.println("World!");  
        }  
    }
```

```
    public void method1() {  
        InnerClass ic = new InnerClass();  
        System.out.println(ic.b);  
    }
```

```
    public static void main(String[] args) {  
        NestedClassExample2 nce2 = new NestedClassExample2();  
        nce2.method1();  
  
        NestedClassExample2.InnerClass ic2 = nce2. new InnerClass();  
        System.out.println(ic2.b);  
        ic2.method2();  
    }  
}
```

Nicht statische Member Klasse

- ❖ Benutzung, wenn innere Klassen eng zur äußeren Klasse gehört.

lokale Klasse

```
public class NestedClassExample3 {  
  
    public void method1() {  
        final int b = 4;  
  
        class LocalClass {  
            int b2 = b;  
            public void method2() {  
                System.out.println("World!");  
            }  
        }  
  
        LocalClass lc = new LocalClass();  
        System.out.println(lc.b2);  
        lc.method2();  
    }  
  
    public static void main(String[] args) {  
        NestedClassExample3 nce3 = new NestedClassExample3();  
        nce3.method1();  
    }  
}
```

Definition
innerhalb einer
Methode

Anonyme Klassen

```
public abstract class AnonymousSuperClass {  
    int a = 3;  
  
    public abstract int method2();  
}
```


Anonyme Klasse

```
public class NestedClassExample4 {  
  
    public void method1(AnonymousSuperClass asc) {  
        int d = asc.method2();  
        System.out.println(d);  
    }  
  
    public static void main(String[] args) {  
        NestedClassExample4 nce4 = new NestedClassExample4();  
        nce4.method1(new AnonymousSuperClass() {  
            int b = 4;  
  
            public int method2() {  
                int c = a * b;  
                return c;  
            }  
        });  
    }  
}
```

Implementierung
der abstrakten
Klasse innerhalb
einer anderen

Anonyme innere Klassen

- ❖ Häufig verwendet bei der Verwendung von Listenern
 - ❖ ActionListener auf Buttons etc.

Im Buch gibt es mehr Infos unter Kapitel 7.1

[http://openbook.galileocomputing.de/javainsel/
javainsel_07_001.html#dodtp0b960ace-1146-4436-b4f6-3f9325e8fd1f](http://openbook.galileocomputing.de/javainsel/javainsel_07_001.html#dodtp0b960ace-1146-4436-b4f6-3f9325e8fd1f)

```
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
```

```
import javax.swing.JButton;
import javax.swing.JFrame;
```

```
public class ButtonEventClass extends JFrame {
```

```
    private JButton button;
```

```
    public ButtonEventClass(){
        ?????auszuimplementierender Code???????
        ?????auszuimplementierender Code???????
    }
```

```
        this.getContentPane().add(button);
    }
```

```
    public static void main(String[] args){
        ButtonEventClass bec = new ButtonEventClass();
        bec.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        bec.setSize(200, 200);
        bec.setVisible(true);
    }
}
```

```
???????auszuimplementierender Code??????????
```

```
}
```


Übung

Ergänzen Sie den fehlenden Code in der Art, dass die Klasse `ButtonEventClass` eine InnerClass `ButtonLauscher` erhält, die die Methode `actionPerformed(ActionEvent e)` des bereits importierten `java.awt.event.ActionListener` implementiert. Die Implementierung soll nichts anderes tun als ausgeben, dass der Button geklickt wurde.

Im Konstruktor der Klasse `ButtonEventClass` ergänzen Sie die Initialisierung des Buttons und weisen dem Button den gerade erstellten `ActionListener` zu. Die Klasse `Button` hält dafür eine Methode `addActionListener` bereit.