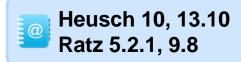




Programmieren II

Innere Klassen



```
Institut für Automation und angewandte Informatik

Ing allResults = new Araing, Integer> typeWordResult, Integer> typePoints = new typePoints);

It is sheets) {

Institut für Automation und angewandte Informatik

Integer> typeWordResult

Integer> typePoints = new Araing

Intege
```

Innere Klassen



- Bisher kennen wir nur Klassen, die entweder zusammen in Paketen organisiert waren oder in einer Datei.
- Diese Form von Klassen heißen "Top-Level-Klassen".
- Es gibt darüber hinaus die Möglichkeit, eine Klasse innnerhalb einer anderen Klasse zu platzieren und sie damit noch enger aneinander zu binden. Eine Klasse, die so eingebunden wird, heißt "innere Klasse". Im Allgemeinen sieht dies wie folgt aus:

```
class OuterClass {
    class InnerClass {
    }
}
```

Innere Klassen - Überblick



- Die Java-Sprache definiert vier Typen von inneren Klassen
 - Statische innere Klassen
 - Mitglieds- oder Elementklassen
 - Lokale Klassen
 - Anonyme innere Klassen

Statische innere Klassen (1)



 Die einfachste Variante einer inneren Klasse wird wie eine statische Eigenschaft (Attribut) in die Klasse eingesetzt.

```
public class Lamp {
    static String s = "Let there be light...";
    int i = 1;
    static class Bulb {
        void shine() {
            System.out.println(s);
            // System.out.println( i );
            // Fehler (da i nicht statisch)
```

Statische innere Klassen (2)



- Die Eigenschaften der statischen inneren Klasse Bulb besitzen Zugriff auf alle statischen Eigenschaften und statischen Methoden der äußeren Klasse Lamp.
- Zugriff auf Objektvariablen ist aus der statischen inneren Klasse nicht möglich, da sie als extra Klasse gezählt wird, die im gleichen Paket liegt.
- Die innere Klasse muss einen anderen Namen als die äußere haben.

Mitglieds- oder Elementklassen (1)



- Eine Mitgliedsklasse (engl. member class), auch Elementklasse genannt, ist ebenfalls vergleichbar mit einem Attribut, nur ist dieses nicht statisch.
- Die innere Klasse kann auf <u>alle</u> Attribute der äußeren Klasse zugreifen. Dazu zählen auch die privaten Eigenschaften.

```
public class Border {
    String s = "curly";

    class Pattern {
        void standard() {
            System.out.println(s);
        }
        // static void always() {
            // Fehler (da innere Klasse Pattern nicht statisch)
        }
}
```

Mitglieds- oder Elementklassen (2)



- Die innere Klasse Pattern hat Zugriff auf alle Eigenschaften von Border.
- Um innerhalb der äußeren Klasse Border eine Instanz von Pattern zu erzeugen, muss ein Objekt der äußeren Klasse Border existieren.
- Das ist eine wichtige Unterscheidung gegenüber den statischen inneren Klassen. Statische innere Klassen existieren auch ohne Objekt der äußeren Klasse.
- Eine zweite wichtige Eigenschaft ist, dass innere Mitgliedsklassen selbst keine statischen Eigenschaften definieren dürfen.

Erzeugung von Objekten innerer Klassen von außen (1)



- Innerhalb der äußeren Klassen kann mit dem new-Operator ein Objekt der inneren Klasse erzeugt werden.
- Kommen wir von außerhalb und wollen Objekte der inneren Klasse erzeugen, so müssen wir bei Elementklassen sicherstellen, dass es ein Objekt der äußeren Klasse gibt. Die Sprache schreibt eine neue Form für die Erzeugung mit new vor, die das allgemeine Format

```
ref.new InnerClass();
```

besitzt.

Dabei ist ref eine Objektreferenz der äußeren Klasse.

Erzeugung von Objekten innerer Klassen von außen (2)



```
class House {
   class Room {
    }
}
```

Um von außen ein Objekt von Room aufzubauen, schreiben wir:

```
House h = new House();
Room r = h.new Room();
```

oder auch in einer Zeile:

```
Room r = new House().new Room();
```

Elementklassen können beliebig geschachtelt sein, und da der Name eindeutig ist, gelangen wir immer mit Classname.this an die jeweilige Eigenschaft.

Erzeugung von Objekten innerer Klassen von außen (3)



```
public class House {
   String s = "House";
   class Room {
      String s = "Room";
      class Chair {
         String s = "Chair";
                                            Ausgabe
         void output() {
            System.out.println(s); -
            System.out.println(Chair.this.s); ---> Chair
            System.out.println(House.this.s); ---> House
   public static void main(String args[]) {
      new House().new Room().new Chair().output();
```

Erzeugung von Objekten innerer Klassen von außen (4)



Objekte für die inneren Klassen House, Room und Chair lassen sich auch wie folgt erstellen:

- Die Qualifizierung mit dem Punkt bei House.Room.Chair bedeutet nicht, dass House ein Paket mit dem Unterpaket Room ist, in dem die Klasse Chair existiert.
- Lesbarkeit erschwert, Verwechslungsgefahr zwischen inneren Klassen und Paketen.
 - Deshalb: Namenskonvention befolgen! Klassennamen beginnen mit Großbuchstaben, Paketnamen mit Kleinbuchstaben.

Lokale Klassen (1)



Lokale Klassen sind innere Klassen, die jedoch nicht als Eigenschaft direkt in einer Klasse eingesetzt werden. Diese Form der inneren Klasse befindet sich in Anweisungsblöcken von Methoden oder Initialisierungsblöcken.

```
public class ItsFunInside {
    public static void main(String args[]) {
         int i = 2;
         final int j = 3;
         class In {
             In() {
                  System.out.println(j);
                  System.out.println(i);
         new In();
                                  Bis Java 8: Fehler; seit Java 8: erlaubt
                               Grund: "effectively final" (ausführlich später bei Java 8)
```

Lokale Klassen (2)



- Die Definition der inneren Klasse In ist wie eine Anweisung eingesetzt.
- Jede lokale Klasse kann auf Methoden der äußeren Klasse zugreifen und zusätzlich auf die lokalen Variablen und Parameter, die mit dem Modifizierer final als unveränderlich ausgezeichnet sind.
- Liegt die innere Klasse in einer statischen Methode, kann sie jedoch keine Objektmethode aufrufen.
- Eine weitere Einschränkung im Vergleich zu den Mitgliedsklassen ist, dass die Modifizierer public, protected, private und static nicht erlaubt sind.

Anonyme innere Klassen (1)



- Anonyme Klassen gehen noch einen Schritt weiter als lokale Klassen. Sie haben keinen Namen und erzeugen immer automatisch ein Objekt.
- Klassendefinition und Objekterzeugung sind zu einem Sprachkonstrukt verbunden. Die allgemeine Notation ist folgende:

```
new KlasseOderSchnittstelle() {
    /* Eigenschaften der inneren Klasse */
};
```

In dem Block geschweifter Klammern lassen sich nun Methoden und Attribute definieren. Hinter new steht der Name einer Klasse oder Schnittstelle (interface).

Anonyme innere Klassen (2)



- Wenn hinter new der Klassenname A steht, dann ist die anonyme Klasse eine Unterklasse von A.
- Wenn hinter new der Name einer Schnittstelle S steht, dann erbt die anonyme Klasse von Object und implementiert die Schnittstelle S.
 Würde sie die Operationen der Schnittstelle nicht implementieren, hätten wir eine abstrakte innere Klasse, von der kein Objekt erzeugt werden könnte (was wir aber genau an dieser Stelle tun).
- Für anonyme innere Klassen sind keine zusätzlichen extends- oder implements-Angaben möglich.





```
import java.awt.Point;
public class InnerToStringPoint {
    public static void main(String args[]) {
        Point p = new Point(10, 12) {
            public String toString() {
                return "(" + x + "," + y + ")";
        };
        System.out.println(p);
                                                       Ausgabe
                                                       (10,12)
```

■ Da sofort eine Unterklasse von Point definiert wird, fehlt der Name der inneren Klasse. Das einzige Exemplar dieser anonymen Klasse lässt sich über die Variable p weiterverwenden.

Anonyme innere Klassen (3)



- Eine anonyme innere Klasse kann nützliche Methoden der Oberklasse überschreiben oder Schnittstellen implementieren.
- Neue Eigenschaften anzubieten wäre zwar legal, aber nicht sinnvoll. Denn von außen wären diese Methoden und Attribute unbekannt, da die zugänglichen Eigenschaften der Oberklasse den Zugriff festlegen.
- Deshalb sind auch anonyme Unterklassen von Object (ohne weitere implementierte Schnittstellen) nur selten nützlich.
- Ein Haupteinsatzgebiet für anonyme innere Klassen ist die Ereignisbehandlung (Event-Handling), z.B. bei Swing.