Einstieg in die Informatik mit Java

Innere Klassen

Gerd Bohlender

Institut für Angewandte und Numerische Mathematik

Gliederung

Gliederung

Einführung

Innere Klassen sind geschachtelte Klassen, d.h. eine Klasse wird in einer anderen deklariert.

Das bringt die folgenden Vorteile mit sich:

- zusammengehörige Klassen werden zusammengefasst,
- die Sichtbarkeit wird kontrolliert,
- Namenskonflikte mit anderen Objekten im gleichem Paket werden vermieden.

Geschachtelte Klassen können unabhängig vom Modifizierer gegenseitig auf ihre Komponenten und Methoden zugreifen.

Gliederung

Element-Klassen

Die innere Klasse wird bei den Elementen der äußeren Klasse deklariert.

- Die Klassennamen müssen sich unterscheiden.
- Die innere Klasse benötigt eine Instanz der äußeren Klasse.
- Der Modifizierer der inneren Klasse ist frei wählbar, d.h. er hängt nicht vom Modifizierer der äußeren Klasse ab.

Syntax

```
class Aussen {
        // Daten und Methoden

        class Innen {
                // Daten und Methoden
        }
}
```

Element-Klassen

Bei der Kompilierung legt der Kompiler für die innere Klasse eine separate `class`-Datei an, im obigen Beispiel:

```
Aussen$Innen.class
```

Achtung

Elementklassen dürfen keine statischen Komponenten, Methoden oder Klassen enthalten, solange die Elementklasse nicht selber `static` ist!

Gliederung

Lokale Klassen

Lokale Klassen werden in einem Block in einer Methode der äußeren Klasse deklariert.

- Die Klassennamen müssen sich unterscheiden.
- Es sind keine Modifizierer sowie statische Komponenten erlaubt.
- Zugriffe auf lokale Konstanten der äußeren Klasse sind erlaubt, dabei müssen die Konstanten allerdings vor der inneren Klasse definiert sein, Zugriffe auf lokale Variablen der äußeren Klasse sind hingegen verboten (Bei der Kompilierung wird eine separate `class`–Datei erzeugt, die eine Kopie der lokalen Variablen enthält.).

Achtung

Ist die Methode statisch, so darf in der inneren Klasse nur auf statische Elemente der äußeren Klasse zugegriffen werden, andernfalls auf alle!

Beispiel

```
class Aussen {
    private int var1;
    public void methode () {
        int var2;
        final int konst1 = 333;
        class Innen {
// innere Klasse
            Innen () {
// Konstruktor
                System.out.println(var1);    // ok
                System.out.println(var2);    // Fehler!
                System.out.println(konst1); // ok
            }
        }
        new Innen();
// Konstruktor-Aufruf
    }
    public static void main(String[] s) {
        Aussen Referenz = new Aussen ();
        Referenz.methode ();
    }
}
```

Gliederung

Anonyme Klassen

Manchmal werden Klassen (Subklassen, Implementierungen einer Schnittstelle) nur für eine einzige Instanz benötigt.

Beispiel

```
...
class MeineKlasse extends Vorlage {
    void info() { ... }  // Methode ueberschreiben
}
MeineKlasse m = new MeineKlasse();
     // einmalig Instanz bilden
...
```

Anonyme Klassen

Dies kann einfacher mit einer anonymen Klasse formuliert werden. Anonyme Klassen sind lokal und verzichten auf einen Bezeichner, also einen Namen für die Klasse.

Beispiel

```
...
Vorlage m = new Vorlage() {
      void info() {
      ...
      }
};
...
```

Es wird eine abgeleitete Klasse der Klasse `Vorlage` gebildet und die Methode `info()` ersetzt. Weiterhin wird eine Instanz der anonymen Klasse erzeugt, die den Typ der Superklasse besitzt.

Anonyme Klassen

- Anonyme Klassen können nur mit Schnittstellen bzw. Basisklassen gebildet werden.
- Definitionen eigener neuer Methoden sind nicht sinnvoll, da diese nicht angesprochen werden können.
- Es kann kein (expliziter) Konstruktor gebildet werden. Damit ist der Zugriff auf den `super`-Konstruktor ebenfalls nicht möglich.
- Es ist insgesamt nur eine Instanz möglich.
- Der Kompiler erzeugt für diese Instanz eine eigene `class`-Datei (an den Namen der äußeren Klasse wird eine fortlaufende Nummer angehängt, also z.B. `Aussen$1.class`).

Achtung

Es sind keine statische Komponenten erlaubt!

Gliederung

`static` geschachtelte Klassen und Schnittstellen

Statisch geschachtelte Klassen bzw. Schnittstellen verhalten sich wie Elementklassen, sind jedoch zusätzlich mit dem Schlüsselwort `static` versehen.

- Sie besitzen keinen Bezug zu einer Instanz und sind daher sogenannte *Top-Level Klassen* bzw. *Top-Level Schnittstellen*.
- Von außen über die Klassennamen *Aussen.Innen* ansprechbar.
- Außer beim Namen besitzt diese Klassengattung keine echte Schachtelung.
- Eine statisch geschachtelte Schnittstelle darf eine Klasse enthalten, welche (implizit) statisch ist und daher eine Top-Level Klasse ist.