3. Inner Classes

- Spracherweiterung in Java 1.1 (Bytecode-Kompatibel mit älteren Versionen)
- Klassen/Interfaces können seit Release 1.1 auch als Elemente einer Klasse definiert werden, wie Felder/Methoden und sogar wie lokale Variablen
- Sinnvolle Verwendung im Zusammenhang mit dem Java Event-Modell (ebenfalls seit 1.1), d.h. für Defnintion von Event-Listener
- Speziell: Innere Klassen haben Zugriff auf Methoden/Felder der Klasse (Methode) in welcher sie deklariert sind, insbesondere auch auf private (!) Felder/Methoden.

Varianten

i. Geschachtelte Statische Klassen
ii. Elementklassen
iii. Lokale Klassen
iv. Anonyme Klassen

3.1. Geschachtelte statische Klassen

```
class Stack {
  static class Node {
    int val;
    Node next;
  }

  Node root;
  ...
}
```

- Deklaration der Klasse Node innerhalb der Klasse Stack
- static (☞ lokale Interfaces sind immer static, auch ohne Angabe von "`static`")
- Klasse definiert eigenen Namensraum, d.h. man ist frei bei der Namenswahl von Klassen welche nur innerhalb einer Klasse benötigt werden.
- Klassenname enthält Name der umliegenden Klasse
  - Stack.class
    Stack$Node.class
- Falls lokale Klasse nicht private deklariert wird kann sie auch aus anderen Klassen verwendet werden:
  - `try { s.pop() } catch (Stack.EmptyExpression e) { … }`
  - `import Stack.Node`
    `import Stack.*;`
- Zugriff auf private (statische) Variablen der umliegenden Klasse möglich

3.2. Elementklassen

- werden ebenfalls innerhalb von anderen Klassen deklariert
- ohne static !
- Eine Elementklasse verweist implizit auf eine Instanz der Klasse in welcher sie definiert ist, und zwar auf jene Instanz, aus welcher die Elementklasse erzeugt wurde.
- Aus einer Elementklasse kann auf die Felder/Methoden der umschliessenden Klasse zugegriffen werden.

Bsp.:

```
class BE extends JComponent {
  private int x, y;

  private class ML extends MouseAdapter {
    public void mouseClicked(MouseEvent e) {
      x = e.getX();
      y = e.getY();
      repaint();
    }
  }

  public BE in () {
    add MouseListener(new ML());
  }

  public void paint(Graphics g) {
    g.setColor(Color.red);
    g.fillRect(x, y, 40, 50);
  }
}
```

Spezialfälle

- expliziter Zugriff auf implizite Referenz (Auflösen von Namenskonflikten od. Zugriff auf äussere Instanz als Ganzes):

  `Klassenname.this` `C.this.x` `C.this.y` `C.this` `C.this.repaint()`

  Regel: Klasse darf nicht denselben Namen wie Umgebungklasse haben.
- `new` darf nur innerhalb der definierenden Klasse aufgerufen werden. Nicht aus statischen Methoden da Instanz verfügbar sein muss.
- Man kann auch explizit eine Instanz auf äussere Klasse übergeben, d.h. explizit Rückreferenz setzen.

3.3. Lokale Klassen

Eine lokale Klasse ist innerhalb eines Codeblocks deklariert
- innerhalb einer Methode (statisch/dynamisch)
- innerhalb eines Initialisierers

Bsp.:
```
class C {
  private int x = 47;
  void createLocal() {
    class Local {
      int n;
      Local(int n) { this.n = n };
      int getX() { return n*x; }
    }
    Local loc = new Local(2);
    System.out.println(loc.getX());  // 94
  }
}
```

Merkmale:
- Zugriff auf Methoden/Instanzen der Klassen in welcher die innere Klasse enthalten ist.
  - Zugriff auf äussere Instanz mit C.this
  - keine äussere Instanz falls innere Klasse innerhalb einer statischen Methode deklariert
- Lokale Klassen sind nur in dem Codeblock sichtbar in welchem sie deklariert sind.
- Zugriff auf lokale Variablen und Parameter

```
class C {
  Data d;
  interface Data { int getX(); }

  Data getData(final int x) {
    class D implements Data {
      public int getX() { return x; }
    }
    return new D();
  }

  static void main(String[] args) {
    C c = new C();
    c.d = c.getData(4);
    System.out.println(c.d.getX();
  }
}
```

Konsequenz:

Lokale Klassen dürfen nur die als final deklarierten Variablen und Methodenparameter des Codestücks verwenden.
Diese Variablen (bei Objekten: Die Referenz selber, nicht das referenzierte Objekt) können sich nicht mehr ändern und können daher kopiert werden.

Lokale Klassen enthalten:
- versteckte Referenz auf äusseres Objekt (falls nicht statisch)
- Kopien aller lokalen finalen Variablen auf welche zugegriffen wird
- Konstruktor angepasst, so dass Referenz nach aussen und Initialwerte der Parameter übergeben werden können.

⇒ Keine Deklaration von statischen Variablen/Methoden möglich in lokalen Klassen.

3.4. Anonyme Klassen

Anonyme Klassen sind lokale Klassen ohne Namen. Die Deklaration und Konstruktion erfolgt in einem. Bsp:

```
class C {
  Data d;
  interface Data { int getX(); }

  Data getData(final int x) {
    return new Data() {
      getX() { return x; }
    }
  }
}
```

Syntax:

Tip: Verwendung anonymer Klassen:

- Falls Klasse sehr klein (nur 1 Methode)
- Falls nur eine Instanz benötigt wird

Bemerkung:

- Anonyme Klassen besitzen keinen Namen
- Anonyme Klassen können keinen Konstruktor haben, aber Konstruktorblöcke