Inner Class

```
1 public class OuterClass {
2     private int var;
3     public class InnerClass {
4         void methodA() {};
5     }
6     public void methodB() {};
7 }
```

- Instanz von `InnerClass` kann auf alle Member von OuterClass zugreifen.
- Wenn `InnerClass` `static` deklariert wird, kann man nur auf statische Member zugreifen.
- Statische innere Klassen sind im Prinzip normale Klassen mit zusätzlichen Zugriffsrechten.
- Nichtstatische innere Klassen sind immer an eine konkrete Instanz der äußeren Klasse gebunden.

Beispiel – Zugriff von Außen

```
1  class OuterClass {
2      private int x = 1;
3      public class InnerClass {
4          void show() {
5              System.out.println("x = " + x);
6      }   }
7      public void showMeth() {
8          InnerClass b = new InnerClass();
9          b.show();
10 }   }
11 public class TestInner {
12     public static void main(String args[]) {
13         OuterClass a = new OuterClass();
14         OuterClass.InnerClass x = a.new InnerClass();
15         x.show();
16         a.showMeth();
17 }   }
```

"TestInner.java"

Beispiel – Zugriff von Außen

```
1  class OuterClass {
2      private static int x = 1;
3      public static class InnerClass {
4          void show() {
5              System.out.println("x = " + x);
6      }   }
7      public void showMeth() {
8          InnerClass b = new InnerClass();
9          b.show();
10 }   }
11 public class TestInnerStatic {
12     public static void main(String args[]) {
13         OuterClass a = new OuterClass();
14         OuterClass.InnerClass x =
15                     new OuterClass.InnerClass();
16         x.show(); a.showMeth();
17 }   }
```

"TestInnerStatic.java"

Local Inner Class

Eine lokale, innere Klasse wird innerhalb einer Methode deklariert:

```
1 public class OuterClass {
2     private int var;
3     public void methodA() {
4         class InnerClass {
5             void methodB() {};
6         }
7     }
8 }
```

- Kann zusätzlich auf die finalen Parameter und Variablen der Methode zugreifen.

Beispiel – Iterator

```java
interface Iterator<T> {
    boolean hasNext();
    T next();
    void remove(); // optional
}
```

- Ein Iterator erlaubt es über die Elemente einer Kollektion zu iterieren.
- Abstrahiert von der Implementierung der Kollektion.
- `hasNext()` testet, ob noch ein Element verfügbar ist.
- `next()` liefert das nächste Element (falls keins verfügbar ist wird eine `NoSuchElementException` geworfen).
- `remove()` entfernt das zuletzt über `next()` zugegriffene Element aus der Kollektion.

Beispiel – Iterator

```
1  public class TestIterator implements Iterable<Integer> {
2      Integer[] arr;
3      TestIterator(int n) {
4          arr = new Integer[n];
5      }
6      public Iterator<Integer> iterator() {
7          class MyIterator implements Iterator<Integer> {
8              int curr = arr.length;
9              public boolean hasNext() { return curr>0;}
10             public Integer next() {
11                 if (curr == 0)
12                     throw new NoSuchElementException();
13                 return arr[--curr];
14             }
15         }
16         return new MyIterator();
17     }
```

"TestIterator.java"

Beispiel – Iterator

Anwendung des Iterators:

```
18      public static void main(String args[]) {
19          TestIterator t = new TestIterator(10);
20          for (Iterator<Integer> iter = t.iterator();
21               iter.hasNext();) {
22              Integer i = iter.next();
23              System.out.println(i);
24          }
25          for (Integer j : t) {
26              System.out.println(j);
27          }
28      }
29  }
```

"TestIterator.java"

In diesem Fall wird nur 20 mal null ausgegeben...