Geneste en anonieme klassen

**DE HOGESCHOOL
MET HET NETWERK**

Hogeschool PXL – Elfde-Liniestraat 24 – B-3500 Hasselt
www.pxl.be - www.pxl.be/facebook

Inhoud

Nested class

Inner class

Local inner class

Anonymous class

Lambda

Functional Interface

Method Reference

Voorkennis

Klassen

Interfaces

static

1 klasse, 1 bestand

new ActionListener() { ... }

Static nested class

```java
public class OuterClass {
    public static class NestedClass {
        ....
    }
}
```

Gebruik (buitenaf):

OuterClass.NestedClass nested = new OuterClass.NestedClass();

Static nested class

```java
public class OuterClass {
    private static int classField = 1;

    public static class NestedClass {
        private int nestedField;
        public NestedClass() {
            nestedField = classField++;
        }
    }
}
```

Static nested class

```java
public class OuterClass {
    private int field = 1;

    public static class NestedClass {
        public void setField(OuterClass outer, int val) {
            outer.field = val;
        }
    }
}
```

Static nested class

```java
public class OuterClass {
    private NestedClass nested;

    public OuterClass() {
        nested = new NestedClass();
    }

    public int getValue() {
        return nested.calculate();
    }

    private static class NestedClass {
        private int calculate() { ... };
    }
}
```

Static nested class: voorbeeld

```java
public class ColorEnums {
    public enum Color {
        RED, GREEN, BLUE, YELLOW, …;
    }
}
```

Gebruik:

ColorEnums.Color color = ColorEnum.Color.RED;

Nested class

Samenvatting:

- Static nested class kan geïnstantieerd worden
- Static nested class (instantie) zit in de Outer class
- private ≈ in hetzelfde bestand

Inner class

```java
public class OuterClass {
    private int field = 1;

    public class InnerClass {
        private int nestedField;
        public InnerClass() {
            nestedField = field;
        }
    }
}
```

Inner class

```java
public class OuterClass {
    public void doSomething() {
        InnerClass inner = new InnerClass();
        inner.doMethod();
    }

    public class InnerClass {
        public void doMethod () { ... }
    }
}
```

Inner class

```java
public class OuterClass {
    public void doSomething() {
        InnerClass inner = this.new InnerClass();
        inner.doMethod();
    }

    public class InnerClass {
        public void doMethod () { ... }
    }
}
```

Inner class

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


OuterClass outer = new OuterClass();
OuterClass.InnerClass inner;
inner = outer.new InnerClass();
```

Inner class

```java
public class OuterClass {
    private int field = 1;

    public class InnerClass {
        private int field;
        public InnerClass(int field) {
            this.field = field;
            OuterClass.this.field = field;
        }
    }
}
```

Local inner class

```java
public class OuterClass {
    public void method() {
        final int CONST = 5;
        int val = 7;

        class LocalInnerClass {
            ...
            System.out.println(CONST); // OK
            System.out.println(val);    // OK (val wijzigt niet).

            ...
        }
        LocalInnerClass local = new LocalInnerClass();
    }
}
```

Local inner class

```java
public class OuterClass {
    public Object getInner() {
        int val = 7;
        class LocalInnerClass {
            public String toString() {
                val++; // Fout
                return "Inner " + val;
            }
        }
        return new LocalInnerClass();
    }
}

public static void main(String[] args) {
    OuterClass outer = new OuterClass();
    outer.getInner().toString();
}
```

Local inner class

```java
public class OuterClass {
    private int val = 1;
    public void method() {
        class LocalInnerClass {
            …
            System.out.println(OuterClass.this.val); // OK
            …
        }
        LocalInnerClass local = new LocalInnerClass();
    }
}
```

Overzicht

Klasse:	In:	Toegang:	Gebruik:
(normal)	package	package/protected/public	Bijna altijd
Static nested	class	+ private	Ten dienst van Outer class
Inner	object	+ Outer.this	Ten dienst van Outer object
Local Inner	method	+ lokale final variabelen	Ten dienst van 1 methode

Anonymous class

```java
public class OuterClass {
    public void method() {
        class SubClass extends SuperClass {
            // Vervangen methodes
        }
        SuperClass object1 = new SubClass();

        class InterfaceClass implements Interface {
            // Implementatie van methodes
        }
        Interface object2 = new InterfaceClass();
    }
}
```

Anonymous class

```
public class OuterClass {
    public void method() {
        SuperClass object1 = new SuperClass() {
            // Vervangen methodes
        };


        Interface object2 = new Interface() {
            // Implementatie van methodes
        };

    }
}
```

Toepassing: iterator

Iterator it;

it = collection.iterator();

while (it.hasnext()) {

    Object o = it.next();

    // use o …

}

for (Object o : collection) {

    // use o …

}

De Iterator interface kan geïmplementeerd worden in een nested class in de collection klasse.

Callback methode

```java
public class Tekst {
    private String sentence;
    …
    public void printFilteredWords(WordFilter filter) {
        for (String word : sentence.split(" ")) {
            if (filter.isValid(word)) {
                System.out.println(word);
            }
        }
    }
}
public interface WordFilter {
    public boolean isValid(String word);
}
```

Callback methode

```java
public class TekstApp {

    ...
    Text text = new Text("Hello this is an example sentence");
    text.printFilteredWords(new WordFilter() {
        @Override
        public boolean isValid(String word) {
            return word.contains("e");
        }
    });
    ...
}
```

Callback methode

```
public class TekstApp {
    ...
    Text text = new Text("Hello this is an example sentence");
    text.printFilteredWords(new WordFilter() {
        @Override
        public boolean isValid(String word) {
            return word.length() > 4;
        }
    });
    ...
}
```

Callback methode

```java
public class TekstApp {

    ...
    Text text = new Text("Hello this is an example sentence");
    text.printFilteredWords(new WordFilter() {
        @Override
        public boolean isValid(String word) {
            return word.startsWith("a");
        }
    });
    ...
}
```

Functional Interface

@FunctionalInterface

```java
public interface WordFilter {
    public boolean isValid(String word);
}
```

Lambda

```java
public class TekstApp {

    …

    Text text = new Text("Hello this is an example sentence");
    text.printFilteredWords(word -> word.contains("e"));

    …
}
```

Lambda

```java
text.printFilteredWords(new WordFilter() {
    @Override
    public boolean isValid(String word) {
        return word.contains("e");
    }
});
```

```java
text.printFilteredWords(

    word ->
        word.contains("e")

);
```

Lambda

text.printFilteredWords(word -> word.contains("e"));

Impliciet:
- new WordFilter() { ... }
  - afgeleid uit type parameter van printFilteredWords
- @Override public ... isValid(...)
  - afgeleid uit @FunctionalInterface WordFilter
- String word
  - afgeleid uit type parameter van isValid
- boolean, return
  - afgeleid uit isValid en de implementatie van één regel

Lambda

```
public class TekstApp {

    …

    Text text = new Text("Hello this is an example sentence");
    text.printFilteredWords(word -> word.length() > 4);

    …

}
```

Lambda

```java
public class TekstApp {

    ...

    Text text = new Text("Hello this is an example sentence");
    text.printFilteredWords(w -> w.startsWith("a"));

    ...
}
```

Lambda

Syntax:

- WordFilter filter = word -> word.charAt(2) == 'i';

- WordFilter filter = word -> {

    return word.charAt(2) == 'i';

  };

- WordFilter filter = (word) -> word.charAt(2) == 'i';

- WordFilter filter = (String word) -> word.charAt(2) == 'i';

Lambda

Voorbeeld:

- WordFilter filter = word -> {

  ```java
      try {
          int value = Integer.parseInt(word);
          return value > 10;
      } catch (NumberFormatException nfe) {
          return false;
      }
  };
  ```

  - Accolades en return zijn nodig bij meerdere statements.

Lambda

Voorbeeld:

- IndexedWordFilter filter = (word, i) -> word.length() <= i + 2;
  - Haken zijn nodig bij meerdere parameters.
  - IndexedWordFilter krijgt de index van het woord in de zin mee
  - Implementatie als extra oefening

Lambda

```
public class TekstApp {
    private String start;

    ...
    Text text = new Text("bob blijft achter de blauwe auto");
    start = "a";
    WordFilter filter = w -> w.startsWith(start);
    start = "b";
    text.printFilteredWords(filter);
    ...
}
```

Lambda

Scope:

- Lambda expressies delen de scope met de omgevende code.

- De namen van parameters mogen niet al in gebruik zijn.

- De laatste waarde geldt, niet de waarde bij aanmaak.

- Er is toegang tot final lokale variabelen.

- This verwijst naar de omgevende klasse.

- Super verwijst naar de superklasse van de omgevende klasse.

Method reference

```java
@FunctionalInterface
public interface WordProcessor {
    public String process(String word);
}


// in Text.java
public void printProcessedWords(WordProcessor processor) {
    for (String word : sentence.split(" ")) {
        System.out.println(processor.process(word));
    }
}
```

Method reference

```java
public interface TextUtil {
    public static String quote(String str) {
        return String.format("<<%s>>", str);
    }
}


// in main:
text.printProcessedWords(w -> TextUtil.qoute(w));
```

Method reference

// in main:

- text.printProcessedWords(w -> TextUtil.qoute(w));

- text.printProcessedWords(TextUtil::qoute);

- String WordProcessor.process(String word)

- String TextUtil.quote(String s)

Method reference

Syntax:

    qualifier::identifier

Mogelijkheden:

val -> ClassName.staticMethod(val)	ClassName::staticMethod
val -> myObject.method(val)	myObject::method
(Type val) -> val.method()	Type::method
val -> new ClassName(val)	ClassName::new

Method reference

data -> TextUtil.quote(data)	TextUtil::quote
woord -> vertaler.vertaal(woord)	vertaler::vertaal
tekst -> System.out.println(tekst)	System.out::println
(String w) -> w.toUpperCase()	String::toUpperCase
(naam, leeftijd) -> new Persoon(naam, leeftijd)	Persoon::new

Method reference

Voorbeelden

data -> show(data)	this::show
getal -> bereken(getal, 5)	/
persoon -> super.toon(persoon)	super::toon
tekst -> tekst.indexOf(",")	/
tekst -> tekst.concat(tekst)	/

Standaard Functional Interfaces

package java.util.function:

- Supplier<T>: T get()

- Function<T, R>: R apply(T t)

- Consumer<T>: void accept(T t)

- Predicate<T>: boolean test(T t)

Standaard Functional Interfaces

```java
public class Text {
    private int i = 0;
    public Supplier<String> split(String teken) {
        i = 0;
        String[] data = text.split(teken);
        return () -> i < data.length ? data[i++] : null;
    }
    …
}
```

Standaard Functional Interfaces

```java
public class TestApp {
    public static void main(String[] args) {
        Text text = new Text("test 1 2 3 test");
        Supplier<String> woorden = text.split(" ");

        String woord = woorden.get();
        while (woord != null) {
            System.out.println(woord);
            woord = woorden.get();
        }
    }
}
```

Standaard Functional Interfaces

```java
public class User {
    public Supplier<String> lees() {
        Scanner lezer = new Scanner(System.in);
        return () -> lezer.next();
    }
    ...
}
```