

Neuerungen in Java 7 & 8

Martin Binder

Easier Numeric Literals

■ Trennzeichen in langen Nummern

```
long i = 100_000_000_000L;
```

■ Binäre Angabe von Nummern

```
int b    = 0b1010__1111;  
int b2   = 0B1101__1010;
```

```
int result = (b & b2);
```

String in Switch

- String-Konstanten können wie Integer und Enums in switches verwendet werden

```
int monthNameToDays(String s) {  
    switch(s) {  
        case "April":      case "June":  
        case "September": case "November":  
            return 30;  
        case "January":    case "March":  
        case "May":         case "July":  
        case "August":     case "December":  
            return 31;  
        case "February":  
            ...  
        default:  
            ...  
    }  
}
```

Diamond Operator

- Erspart das redundante doppelt-deklarieren von Typinformationen

```
List<String> list = new ArrayList<String>  
=  
List<String> list = new ArrayList<>
```

```
List<Map<String, List<String>>> list = new ArrayList<Map<String, List<String>>>  
=  
List<Map<String, List<String>>> list = new ArrayList<>
```

Multicatch

- Mehrere Exceptions können im gleichen Catch-Block gefangen werden
- Innerhalb der Catch-Methode wird die Basis-Klasse Exception verwendet

```
try
{
    ...
}
catch (final NotAvailableException | NotFoundException | IntegrationException e)
{
    sLog.error("Could not set database hostname to '" + device.getHostname()
        + "' on device '" + device + '\'', e);
}
```

Automatic Resource Management

- Anforderung: Klasse im try implementiert das Interface AutoClosable
- AutoClosable ist Super-Interface von java.io.Closable → Alle bisherigen IO-Klassen sind Auto-Ressource ready
- Bei mehreren Exceptions innerhalb eines try-blocks werden Folge-Exceptions „Suppressed“, sind aber noch im Stacktrace zu sehen

```
try(InputStream in = new FileInputStream(src);  
    OutputStream out = new FileOutputStream(dest)) {  
    // ...  
}
```

New I/O - Die wichtigen Klassen

■ Path

- Repräsentiert einen Pfad im Filesystem, ersetzt `java.io.File`
- Wird erzeugt mit `Paths.get(String)`

■ Files

- Statische Methoden zum Arbeiten mit Dateien und Ordnern

■ Filesystem

- Kern der Architektur
- Repräsentiert das Filesystem auf dem gearbeitet wird
- `Filesystem.getDefault()` gibt das Filesystem des Betriebssystems zurück
- Kann beliebig ausgetauscht werden. Filesystem-Operationen per HTTP? Kein Problem!
- Java 7 liefert Filesystem-Provider für ZIP/Jar mit

New I/O - Files

- Methoden werfen IO-Exception, keine booleans mehr!

- Wichtige Methoden

- Erzeugen: `createFile() / createDirectory()`
- Delete on exit(): `createTempFile() / createTempDirectory()`
- Streams: `newInputStream() / newOutputStream() / newSeekableByteChannel()`

- Deutlich schneller als in Java6:

- Kopieren: `copy() / move()`
- Visitor-Pattern: `walkFileTree()`
- DirStream mit Wildcards: `newDirectoryStream()`

Lambdas


```
interface OhneParameter {  
    void execute();  
}
```

Vor Java 8:

```
OhneParameter ohneParameter = new OhneParameter() {  
    public void execute() {  
        System.out.println("execute()");  
    }  
};  
ohneParameter.execute();
```

Mit Java 8:

Funktioniert mit jedem Interface und jeder abstrakten Klasse mit einer Methode



```
OhneParameter lambda = () -> System.out.println("execute()");  
lambda.execute();
```

Lambdas mit Parameter

```
interface MitParameter {  
    void execute(int a, String b);  
}
```

Vor Java 8:

```
MitParameter mitParameter = new MitParameter() {  
    public void execute(int a, String b) {  
        System.out.println("a: " + a + ", b: " + b);  
    }  
};
```

```
mitParameter.execute(1, "Hello");
```

Mit Java 8:

```
MitParameter lambda = (a, b) ->  
    System.out.println("a: " + a + ", b: " + b);
```

```
lambda.execute(1, "Hello");
```

Typen sind unnötig



Lambdas mit Rückgabewert

```
interface MitRueckgabewert {  
    int execute();  
}
```


Vor Java 8:

```
MitRueckgabewert mitRueckgabewert = new MitRueckgabewert() {  
    public int execute() {  
        return 42;  
    }  
};
```

```
System.out.println(mitRueckgabewert.execute());
```

Mit Java 8:

```
MitRueckgabewert lambda = () -> 42;
```

return unnötig


```
System.out.println(lambda.execute());
```

Lambdas mit mehr als einer Zeile Code


```
interface MitVielCode {  
    int execute();  
}
```

Vor Java 8:

```
MitVielCode mitVielCode = new MitVielCode() {  
    public int execute() {  
        System.out.println("1");  
        return 42;  
    }  
};
```

```
System.out.println(mitVielCode.execute());
```

Mit Java 8:

```
MitVielCode lambda = () -> {  
    System.out.println("1");  
    return 42;  return nötig  
};
```

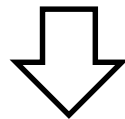
```
System.out.println(lambda.execute());
```

Methodenreferenzen

```
interface MitString{  
    void execute(String a);  
}
```

```
MitString lambda = (a) -> System.out.println(a);
```

```
lambda.execute("Hello, World");
```



Method reference operator

```
MitString reference = System.out::println;
```

```
reference.execute("Hello, World");
```

Zeigt auf System.out.println(String)

Quiz: Welche Methode wird aufgerufen?

```
interface Reference {  
    void doSomething(int x);  
}
```

```
public void method(String a) {  
    System.out.println("method(String)");  
}
```

```
public void method(int a) {  
    System.out.println("method(int)");  
}
```

```
public void method(boolean a) {  
    System.out.println("method(boolean)");  
}
```

Welche Methode wird aufgerufen?

```
Reference reference = ReferenceQuiz::method;  
  
reference.doSomething(...);
```