Annotationen

Carsten Gips (HSBI)

Was passiert hier?

```java
public class A {
    public String getInfo() { return "Klasse A"; }
}

public class B extends A {
    public String getInfo(String s) { return s + "Klasse B"; }

    public static void main(String[] args) {
        B s = new B();
        System.out.println(s.getInfo("Info: "));
    }
}
```

Was passiert hier?

```java
public class A {
    public String getInfo() { return "Klasse A"; }
}

public class B extends A {
    public String getInfo(String s) { return s + "Klasse B"; }

    public static void main(String[] args) {
        B s = new B();
        System.out.println(s.getInfo("Info: "));
    }
}
```

Hilft `@Override`?

Annotationen: Metadaten für Dritte

- **Zusatzinformationen** für Tools, Bibliotheken, ...
- Kein direkter Einfluss auf die Ausführung des annotierten Codes

- Beispiele:
  - Compiler (JDK): `@Override`, `@Deprecated`, ...
  - Javadoc: `@author`, `@version`, `@see`, `@param`, `@return`, ...
  - JUnit: `@Test`, `@Before`, `@BeforeClass`, `@After`, `@AfterClass`
  - IntelliJ: `@NotNull`, `@Nullable`
  - Checker Framework: `@NonNull`, `@Nullable`, ...
  - Project Lombok: `@Getter`, `@Setter`, `@NonNull`, ...
  - Webservices: `@WebService`, `@WebMethod`
  - ...

Demo: annotations.B: @Override, @Deprecated

Dokumentation mit Javadoc

```java
/**
 * Beschreibung Beschreibung Beschreibung
 *
 * @param date Tag, Wert zw. 1 .. 31
 * @return true, falls Datum gesetzt wurde; false sonst
 * @see java.util.Calendar
 * @deprecated As of JDK version 1.1
 */
public boolean setDate(int date) {
    setField(Calendar.DATE, date);
}
```

@NotNull mit IntelliJ

```java
/* o should not be null */
public void bar(Object o) {
    int i;
    if (o != null) {
        i = o.hashCode();
    }
}
```

@NotNull mit IntelliJ

```java
/* o should not be null */
public void bar(Object o) {
    int i;
    if (o != null) {
        i = o.hashCode();
    }
}
```

```java
/* o must not be null */
public void foo(@NotNull Object o) {
    // assert(o != null);  //  Wirkung (von IntelliJ eingefügt)
    int i = o.hashCode();
}
```

Demo: annotations.WuppieAnnotation: @NotNull

Eigene Annotationen erstellen

```java
public @interface MyFirstAnnotation {}

public @interface MyThirdAnnotation {
    String author();

    int vl() default 1;
}

@MyFirstAnnotation
@MyThirdAnnotation(author = "Carsten Gips", vl = 3)
public class C {}
```

Demo: annotations.C

Annotationen bearbeiten: Java Annotation-Prozessoren

```java
@SupportedAnnotationTypes("annotations.MySecondAnnotation")
@SupportedSourceVersion(SourceVersion.RELEASE_17)
public class Foo extends AbstractProcessor {
    @Override
    public boolean process(Set<? extends TypeElement> as, RoundEnvironment re) {
        for (TypeElement annot : as) {
            for (Element el : re.getElementsAnnotatedWith(annot)) {
                processingEnv.getMessager().printMessage(Diagnostic.Kind.NOTE,
                    "found @MySecondAnnotation at " + el);
            }
        }
        return true;
    }
}
```

Demo: annotations.C und annotations.Foo, META-INF

Wrap-Up

- Annotationen: Metadaten zum Programm
- Typische Anwendungen: Compiler-Hinweise, Javadoc, Tests

- Annotationen können auf Deklarationen (Klassen, Felder, Methoden) angewendet werden
- Annotationen können relativ einfach selbst erstellt werden
  - Definition fast wie ein Interface
  - Einstellung der Sichtbarkeit und Verwendbarkeit und Dokumentation über Meta-Annotationen
- Verarbeitung von Annotationen zur Compilier-Zeit mit Annotation-Processor
- Verarbeitung von Annotationen zur Laufzeit mit Reflection (siehe spätere VL)

LICENSE