

# Objektorientierte Programmierung


Hochschule Bochum

WS 19/20

Dr.-Ing. Darius Malysiak

# III. Vertiefung Java 1-8

## Annotations (Annotationen):

- Erlaubt Einbindung von Metadaten in Sourcecode.
- Seit Java 1.5 verfügbar (JSR-175).
- Stellt eine signifikante Erweiterung der Sprache Java dar.
- Annotationen sind unabhängig von Reflections!  Später mehr dazu.

```
public class Container {  
    int value;  
    @Deprecated  
    public void setValue(int val)  
    {  
        value = val;  
    }  
}
```

# III. Vertiefung Java 1-8

## Annotations:

```
public class Container {  
    int value;  
    @Deprecated  
    public void setValue(int val)  
    {  
        value = val;  
    }  
}
```

```
public class Main {  
    public static void main(String[] args) {  
        Container c = new Container();  
        c.setValue(20);  
    }  
}
```



Warning:(12, 10) java: setValue(int) in  
solutions.exercise4.Container has been deprecated

Mit Hilfe der Annotation ,@Deprecated' wird ein Element als ,veraltet' deklariert. Der Compiler gibt eine Warnung hierzu aus.

# III. Vertiefung Java 1-8

## **Annotations:**

Anwendungen:

- Augmentieren von Sourcecode via Reflection.
  - Neue Methoden hinzufügen.
  - Neue Klassen hinzufügen.
  - Automatisierte Dokumentation.
  - Initialisierung Variablen.
- Bereitstellen von Metainformationen zur Absicherung des Kompiliervorgangs (z.B. API Sicherheit).
- Konfiguration von anderen Annotationen.
- Reduktion von Boilerplate Code.

# III. Vertiefung Java 1-8

## Annotations:

@Override

### Version A

```
public class Container {  
    public void implementation()  
    {}  
}
```

```
public class ContainerChild extends Container{  
    public void implementation()  
    {}  
}
```



Äquivalent hinsichtlich der Business Logic.

### Version B

```
public class Container {  
    public void implementation()  
    {}  
}
```

```
public class ContainerChild extends Container{  
    @Override  
    public void implementation()  
    {}  
}
```

# III. Vertiefung Java 1-8

## Annotations:

@Override

☺ API Change ☺

```
public class Container {  
    public void implementation()  
    {}  
}
```



```
public class Container {  
}
```

Version A



Version B



Error:(5, 5) java: method does not override or implement a method from a supertype

# III. Vertiefung Java 1-8

## Annotations:

Default Annotations (in Java inkludiert)

- `@SuppressWarnings`: z.B. `@SuppressWarnings({"deprecation"})`
- `@Deprecated`
- `@Override`
- `@Retention`
- `@Target`
- ...



Innerhalb neuer Annotationen, später mehr!

# III. Vertiefung Java 1-8

## Reflection / Introspection:

Reflection:

Die Fähigkeit ein Objekt zur Laufzeit auf Java Sprachelemente abzubilden wird als Reflection / Introspection bezeichnet.

Java Object

```
public class Container {  
    int value;  
    @Deprecated  
    public void setValue(int val)  
    {  
        value = val;  
    }  
}
```



Class: Container  
Constructors: ,Container'  
Members: ,value', ,setValue'  
Modifiers: value:protected, ...  
...



# III. Vertiefung Java 1-8

## Reflection / Introspection:

```
try
{
    Class clazz = Class.forName("solutions.exercise4.Container");
    Object object = clazz.newInstance();
    Field[] fields = clazz.getDeclaredFields();
    System.out.println("Fields");
    for( Field f : fields )
    {
        f.setAccessible(true);
        f.set(object, 11);
        System.out.println("name: "+f.getName()+"\t"
            +"type: "+ f.getType().getName()+"\t"
            +"value: "+ f.get(object));
    }
}
catch( Exception e )
{
    e.printStackTrace();
}
```

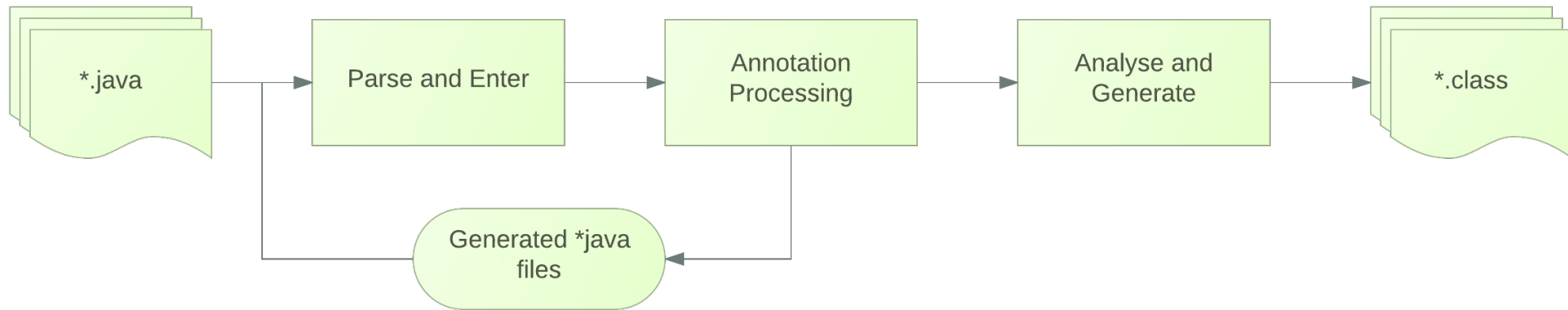


Fields  
name: value type: int value: 11

# III. Vertiefung Java 1-8

## Annotations:

Erstellen eigener Annotationen:



<https://raw.githubusercontent.com/igvalle/igvalle.github.io/master/public/images/butterknife-java-compiler.png>

# III. Vertiefung Java 1-8

## Annotations:

Erstellen eigener Annotationen:

```
@Target(ElementType.METHOD)  
@Retention(RetentionPolicy.SOURCE)  
public @interface Builder {  
}
```

Annotation darf nur an Methoden geschrieben werden.



Annotation wird vor dem Kompilieren entfernt.



```
@SupportedAnnotationTypes("de.dmalysiak.utilities.annotations.Builder")  
@SupportedSourceVersion(SourceVersion.RELEASE_8)  
@AutoService(Processor.class)  
public class Builder extends AbstractProcessor {  
  
    @Override  
    public boolean process(Set<? extends TypeElement> annotations, RoundEnvironment roundEnv) {  
        for (TypeElement annotation : annotations) {...}  
    }  
}
```

# III. Vertiefung Java 1-8

## Annotations:

Erstellen eigener Annotationen:

```
public class Entity {  
    private int intValue1;  
  
    @Builder  
    public void setIntValue1(int val)  
    {  
        intValue1 = val;  
    }  
}
```



Compiler erstellt neue Klasse!

```
public class EntityBuilder {  
    private Entity object = new Entity();  
  
    private EntityBuilder() {}  
  
    public static EntityBuilder get() {  
        return new EntityBuilder();  
    }  
  
    public Entity build() {  
        return object;  
    }  
  
    public EntityBuilder setIntValue1(int value) {  
        object.setIntValue1(value);  
        return this;  
    }  
}
```

# III. Vertiefung Java 1-8

## Annotations:

Erstellen eigener Annotationen:

```
public class Entity {  
    private int intValue1;  
  
    @Builder  
    public void setIntValue1(int val)  
    {  
        intValue1 = val;  
    }  
}
```

Test ☺ (später mehr)

```
@Test  
public void testCreation() {  
    Entity e =  
        EntityBuilder.get().setIntValue1(1).build();  
  
    assertTrue( e.getIntValue1() == 1 );  
}
```

Builder Pattern

# III. Vertiefung Java 1-8

## Annotations:

Beispiel: **Lombok** Bibliothek

```
@RequiredArgsConstructor  
public class Entity {  
    @Getter  
    @Setter  
    @NonNull  
    private Integer intValue1;  
}
```



Erstellt folgende Elemente in der Klasse ,Entity':

- Konstruktor mit genau einem Parameter für ,intValue'.
- Prüft im Konstruktor ob der Parameter null ist, wirft NPE falls zutreffend.
- Setzt den Modifier des default-Konstruktors auf private.
- Fügt Getter und Setter Methoden für ,intValue' hinzu.

Häufig werden neue Annotationen über 3rd Party Module bereitgestellt.

# III. Vertiefung Java 1-8

## Annotations:

Beispiel: **Lombok** Bibliothek

Moment:



1. Ein Annotation Processor erstellt neue Java Dateien / Klassen.
2. Reflection ist ein Laufzeit Feature!
3. Wie können neue Methoden vom Compiler hinzugefügt werden? ?

**Modifikation vom Abstract Syntax Tree (AST)**

# III. Vertiefung Java 1-8

## Tests:

Wie kann sichergestellt werden, dass eine Softwarekomponente wie gewünscht funktioniert?

- Viele Entwickler arbeiten an der Code Basis.  Automatisierte Tests (wird nach einem Merge kompiliert?).
- Schnittstellen sind hinsichtlich der Signatur (sprachliche Semantik) UND der Daten (fachliche Semantik) zu betrachten.  
 Fachliche Anforderungen in Tests einbetten (z.B. durch Integrationstests).
- Über lange Zeit werden viele Komplexe Abhängigkeiten zwischen Schnittstellen aufgebaut.

 Atomare Tests mit Mocks zwischen Ebenen

 Später mehr. 



# III. Vertiefung Java 1-8

## Tests:

Mögliche Module / Frameworks für Tests:

- Junit
- JTest
- Mockito
- Powermock
- ...

```
import org.junit.Test;
```

```
public class TestClass {
```

```
    @Test
```

```
    public void shouldFailIfWrongValue()
```

```
    {
```

```
        Container c = new Container();
```

```
        c.setValue(11);
```

```
        assert(c.getValue() == 11);
```

```
    }
```

```
}
```

Selbsterklärender Name



Test endet mit Assertion



# III. Vertiefung Java 1-8

## Tests:

Unit Tests VS Integration Tests:

- Unit Tests prüfen ob ‚kleine‘ / elementare Bestandteile einer Komponente funktionieren.
- Unit Tests haben keine Interaktion mit externen Systemen / Komponenten.
- Unit Tests haben eine geringe Laufzeitkomplexität, treffen jedoch keine Aussage über die Integration der zu testenden Komponente in die Code Basis / Applikation. Darüber hinaus werden keine / wenige Wechselwirkungen mit weiteren Bestandteilen der gleichen Komponente getestet.
- Integration Tests arbeiten auf einer Sammlung von (externen) Komponenten um die Wechselwirkung mit der zu testenden Komponente zu prüfen.
- Häufig werden komplexe elementare Bestandteile im Rahmen eines Integration Tests geprüft.
- Integration Tests sind sehr Zeitaufwändig im Vergleich zu Unit Tests.

# III. Vertiefung Java 1-8

## Tests:

Unit Tests VS Integration Tests:

```
public class DatabaseConnector {  
    public void connectDatabase() {...}  
    public void save(Container c) {...}  
    public boolean checkIfSaved(Container c) {...}  
    ...  
}
```

Wie soll der Test für Container aussehen? Was soll wie getestet werden?

```
public class Container {  
  
    public void saveToDatabase(DatabaseConnector connector)  
    {  
        connector.connectDatabase();  
        connector.save(this);  
    }  
}
```

# III. Vertiefung Java 1-8

## Tests:

- Die Klasse ‚DatabaseConnector‘ sollte über separate Tests abgedeckt werden.
- Die Aufrufe für ‚connectDatabase‘ und ‚save‘ können simuliert werden.
- ‚saveToDatabase‘ hat keinen Rückgabewert. ... .. ?
- Evtl. sollte ‚saveToDatabase‘ einen Wert zurückgeben.

Refactoring ☹



```
public void saveToDatabase(DatabaseConnector connector)
{
    connector.connectDatabase();
    connector.save(this);
}
```

# III. Vertiefung Java 1-8

## Tests:

```
public void saveToDatabase(DatabaseConnector connector)
{
    connector.connectDatabase();
    connector.save(this);
}
```



Muss ,checkIfSaved' ebenfalls geprüft werden?

```
public boolean saveToDatabase(DatabaseConnector connector)
{
    connector.connectDatabase();
    connector.save(this);
    boolean result=checkIfSaved(this);
    return result;
}
```

# III. Vertiefung Java 1-8

## Tests:

Mocks / Mocking (Simulation von Komponenten):

- Wie können die Aufrufe für ‚connectDatabase‘ und ‚save‘ simuliert werden?

```
public class DatabaseConnectorStub extends DatabaseConnector{  
    public void connectDatabase() {return;}  
    public void save(Container c) {return;}  
    public boolean checkIfSaved(Container c) {return true;}  
}
```

```
public void shouldFailIfNotSaved()  
{  
    Container c = new Container();  
    DatabaseConnector connector = new DatabaseConnectorStub();  
    assert (c.saveToDatabase(connector)==true );  
}
```

# III. Vertiefung Java 1-8

## Tests:

Was sollte zuerst da sein: Tests oder Business Logic?



Test Driven Development (TDD)

Modul im Master Studiengang