KARLSRUHER ENTWICKLERTAG

Besseren Java-Code mit Type Annotations

Dragan Zuvic 2016
@dzuvic
w11k / theCodeCampus

Überblick

About Me

JSR 308

<> Motivation

(Typ) Annotationen

Checker Framework

Einbinden </>

Fazit

<> Probleme

Vorteile

Fragen & Referenzen </>

Dragan Zuvic

<> **Dragan Zuvic:**

- Full-Stack Entwickler
- + weitere Rollen
- mit Java / Scala / TypeScript
- @dzuvic

Über uns

<> Gegründet 2000
Entwicklung / Consulting
Web / Java / Scala Projekte </>

---

thecodecampus</>

<> Gegründet 2013
Schulungen (seit 2007)
Projekt-Kickoffs </>

Motivation

Wo Typ-Systeme geholfen hätte

<> **Maßeinheiten**

- 1983: Air Canada (Gewicht)
- 1999: Mars Climate Orbiter (Kraft)
- 2003: Hochrheinbrücke Laufenburg (0-Pegel)

Typ System

<> **Was ist ein Typsystem?**

- A type system is a tractable syntactic method for proving the absence of certain program behaviors by classifying phrases according to the kinds of values they compute.
- ≈ System der Zusicherung

<> **Vorteile in der Java Entwicklung**

1. Verifikation
2. Dokumentation
3. Tool-Unterstützung

Ziel von JSR 308

<> **Idee: Typ-System als Plug-In**

- Gilad Bracha: Pluggable Type Systems (2003)
- Optionale Systeme: TypeScript, Groovy, Dart, Strongtalk
- Plug-In Typ Systeme → Metadaten überall (AST)

<> **Optionale Type Qualifier**

`Integer laenge=10` **vs.** `@m Integer laenge=10 * m`

<> - Verifikation: Individuell & Zuschaltbar

- Noch bessere **Dokumentation** →Besseres Verständnis

Java Typ System

Expert

<> **Taxonomie Typ-Systeme?**

- **Static**, Dynamic, **Typ-Sicher**, **Nominell**, Strukturell, *Plug-In Typen*, Union, Intersection, Dependent, Entscheidbar? ...

<> **Wiederholung: Typen in Java**

- Primitives: *IntegralType, FloatingPointType, boolean*
- Refrence Types: ClassType, InterfaceType, TypeVariable, ArrayType
- Ohne Namen: null Type JLS 3.10.7
- ~~void~~ JLS 14.8

Annotations

Java Annotations

<> Deklarative Annotationen

- **Java:** v > 1.4 ∧ v < 1.8
- **Metadaten:** Quelltext ⟷ Javadoc (xdoclet)
- **Retention:** Source, Class, Runtime
- **Target:** type, field, method, parameter, constructor, local_variable, annotation_type, package
- Nur an Deklarationen
- **Verarbeitung:** Introspection, JSR269 oder apt

Breite Akzeptanz: JUnit, TestNG, Hibernate, CDI, EJB, JAX-RS, JAX-B, Spring, Guice, Findbugs, Lombok ...

Type Annotations

<> **Type Annotations**

- Seit Java 8: An Typen
- JSR 308: Syntax & Bytecode
- In JDK 8: Keine Built-In Annotations
- Neue Targets: **type_parameter, type_use**
- Verarbeitung: JSR 269 & *Compiler API* (com.sun.*)
- Reflection unsichtbar

Risiken Umsetzung JSR 308

> Risk: That the broad Java development community is not interested in developing or using pluggable type systems.
>
> —JEP 104

Type Annotations Syntax

<> **Syntax**

- A type annotation appears before the type's simple name
- on a wildcard type argument [..] before the wildcard
- [..] given array level prefixes the brackets [..]
- [..] in front of a constructor declaration [..]
- [..] explicitly declare the method receiver as the first formal parameter [..]
- [..] write an annotation on a type parameter declaration. [..]

Invalide Type Annotations Syntax

<> Eigentlich überall an Typen außer:

```
1  // Annotationen
2  @Annotation4TypeUse @TypeUse Object o;
3
4  //.class
5  @TypeUse Integer.class
6
7  // Import
8  import @TypeUse java.util.List
9
10 // static access (Clazz = scoping)
11 @TypeUse Clazz.staticMember
12
13 // super reference
14 @TypeUse TASyntax.super.memberName
```

Type Annotations Syntax Beispiele

Demo

```
1  public class TASyntax<@TypeParam T extends @TypeUse Number>
2      implements @TypeParam Virtual  {
3      Map<@TypeUse String, @TypeUse List<@TypeUse Document>> xmlDocument;
4      Collection<@TypeParam ? super @TypeUse Document> contractCollection;
5      String @TypeUse [] typedArray;
6      public void whereToPut () throws @TypeUse IOException {
7          @TypeParam Car c = new @TypeParam DirtyCar();
8          DirtyCar d = (@TypeUse DirtyCar & Virtual) c;
9          boolean b = d instanceof @TypeUse Car;
10     }
11     public static final TASyntax<Integer> newone() {
12         return new <Integer>@TypeUse TASyntax();
13     }
14     public <@TypeParam X extends T> void specialBound ( X x ) {} ;
15 }
```

Checker Framework

Checker Framework

<> **Umfang**
- Reihe von **Typ-Annotationen**: NonNull ...
- Einzeln nutzbare **Typ Checker**
- diese bauen auf: **Checker Framework**
- **Wrapper** für javac
- **Annotierte Bibliotheken**: guava, rt.jar,..
- Handbuch, Beispiele, Eclipse-Plugin,...

<> **Unterstützung: Java 8 & Java 7**

```
1  /* @Nonull */ Integer laenge;
```

<> **Aus dem universitären Umfeld (seit 2007)**
- Michael Ernst - University of Washington
- Werner M. Dietl - University of Waterloo

Der UnitChecker

Demo

```
1  static final @m int m = new @m Integer(1);
2  static final @s int s = new @s Integer(1);
3
4  void demo() {
5    @m int laenge = 5 * m;
6    @s double zeit = 1 * s;
7
8    @mPERs double mProS = laenge / zeit;
9  // Fails
10 //  @kmPERh double kmPERhdouble = laenge / zeit;
11 }
```

Der String Format Checker

Demo

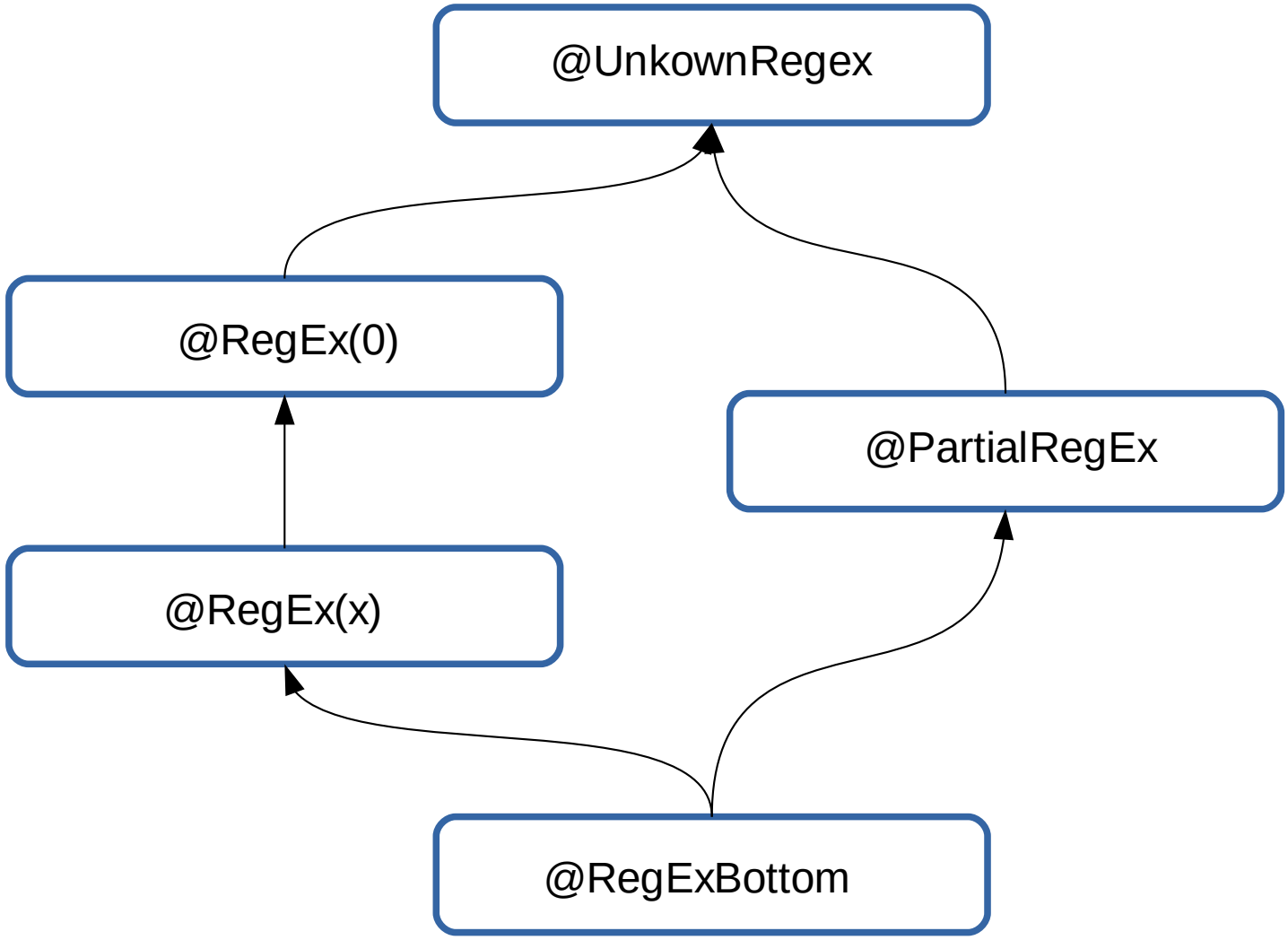
```
1  String str = "str";
2  @Nullable String strN = null;
3
4  String format="%s";
5  String twoformat="%s %f";
6
7  String s1 = String.format("%s", str);
8  String s_checkedBy = formatMe(twoformat, str, 1.0f);
9
10 //         String s_errorNull = String.format("%s", strN);
11 //         String s_errorWrongType = String.format("%d", "zero.one");
```

Der Regular Expression Checker

Demo

```
1  public final static String parenOpen ="(";
2
3  public void doSometing() {
4    String thePattern = "[a-z]+";
5
6  // @RegEx is inferred, so the following would not compile
7  //          String thePattern = "invalidPattern" + parenOpen;
8    Pattern compiledPattern = Pattern.compile( thePattern );
9  }
```

Regex Typ Hierarchie



Nullness Checker

<> **@NonNull @Nullable @MonotonicNonNull**

- ⊘De-Referenzierung von Nicht @NonNull Typen
- ⊘Belegen von null bei @NonNull Typen

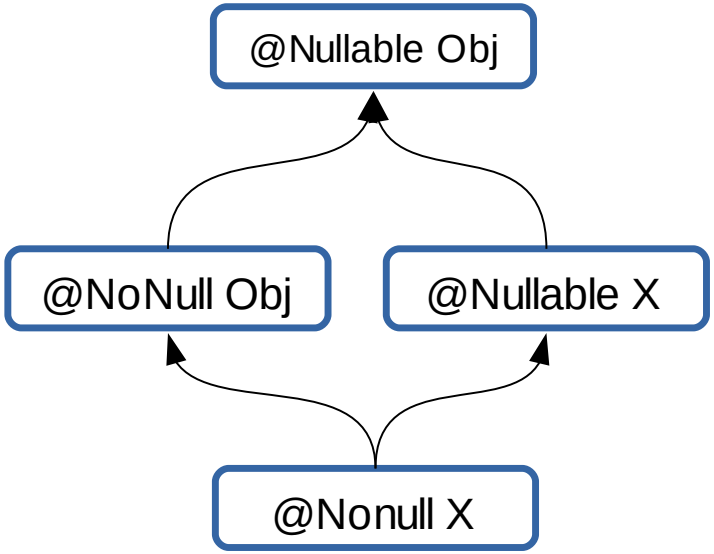
<> **Nullness Checker "aktiviert" noch:**

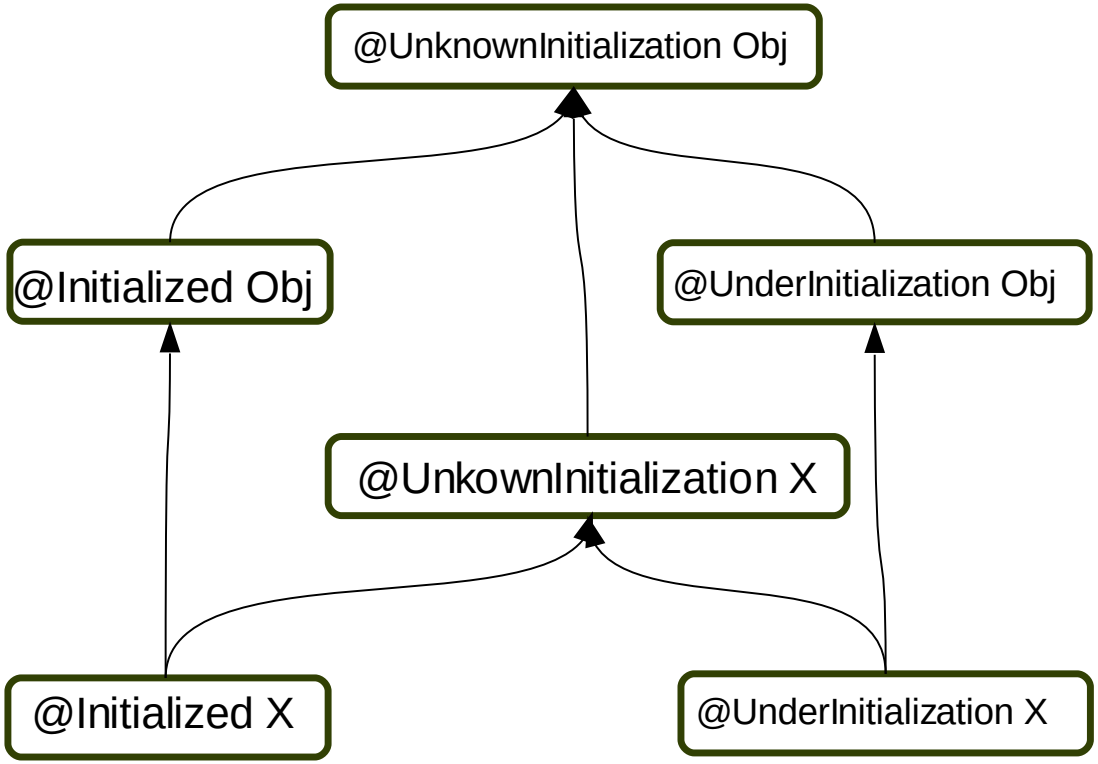
- *Initilization Checker*
  Prüft Zugriff nicht initialisierten Code (NPE)

- *Rawness Initialization Checker*
  Alternative Prüfung der Initialisierung

- *Map Key Checker*
  Prüft: Map.containsKey(key)==true

Nullness Checker Hierarchie





Nullness Checker Einfaches Beispiel

Demo

```
1  public class NonNullInstance {
2    final @NonNull Object nn;
3    final List<@NonNull String> list = new ArrayList<String>();
4
5    public NonNullInstance(@NonNull Object nn, String ... entries) {
6      this.nn = nn;
7      this.list.addAll(Arrays.asList(entries));
8    }
9  }
```

Folgendes compiliert nicht:

```
1  NonNullInstance x = new NonNullInstance(null);
2  NonNullInstance y = new NonNullInstance("some", null, null);
3
```

Nullness Checker - @EnsuresNonNullIf

Demo

<> @NonNull / @Nullable abhängig vom Aufruf

```
1  public class QueueDelegate {
2    private final @NonNull Queue<@NonNull Integer> queue = new LinkedList<>();
3
4    @EnsuresNonNullIf(expression="pop()", result=false)
5    public boolean isEmpty() {return  this.queue.isEmpty();}
6
7    public Integer pop() {
8      return this.queue.isEmpty() ? null :  this.queue.poll();
9    }
10 }
11 /* ... */
12 QueueDelegate dlg = new QueueDelegate();
13 @NonNull Integer doesCompile = dlg.isEmpty() ? 0 : dlg.pop();
14 //@NonNull Integer doesNotCompile = dlg.pop();
```

Nullness Checker Advanced

<> @MonotonicNonNull

<> @PolyNull

<> Inferenz `(x != null)`

<> Externe Tools für Inferenz Annotation

<> `System.getProperty("java.version")` ist @NonNull !

<> Arrays

- `@NonNull Integer @Nullable []`
- `@NonNull String[] x = new @NonNull String[10]; // FEHLER`

Interning Checker

Expert

<> **Interning** → `x.equals(y) ≡ x==y`

- Andere Beispiele: Integer; Read-Only Objekte
- Interning bspw. mit `WeakHashMap` → Guava Cache

```
1 @Interned String s1 = "10";
2 @Interned String s1_same = intern(new String("10"));
3 // @Interned String s1_same = new String("10");
4
5 if (s1 == s1_same) System.out.println("same string");
```

Tainting Checker

Expert

<> **Tainting: Typ-Übergang** `@Tainted` → `@Untainted`

- SQL-Injection, Passwort Hashing, Verschlüsselung Kreditkarten

```
1  @Tainted @Nullable String pass = console.readLine("Password:");
2  @Untainted String unTaintedPassword = hashPassword(pass);
3  boolean passwordCorrect = checkPassword(unTaintedPassword);
4  //...
5  @NonNull boolean checkPassword(@Untainted @Nullable String pass) { .. }
6  //...
7  @Untainted
8  @NonNull
9  public String hashPassword(@Nullable String pass) { ...
10   @SuppressWarnings({"tainting"})
11   @Untainted String result = Base64.getEncoder().encodeToString(res);
12   return result;
13 }
```

Subytpe Checker

Expert

<> Eigene Typ-Hierarchien definieren:

```
1- @SubtypeOf(PossiblyUnencrypted.class)
2  @Target({ElementType.TYPE_USE, ElementType.TYPE_PARAMETER})
3  public @interface Encrypted {}
```

```
1- @SubtypeOf({})
2  @DefaultQualifierInHierarchy
3  @Target({ElementType.TYPE_USE, ElementType.TYPE_PARAMETER})
4  public @interface PossiblyUnencrypted {}
```

<> Typ-Hierarchie als Parameter:

```
-Aquals=Encrypted,PossiblyUnencrypted,PolyEncrypted
```

```
<> // does not compile
2  @Encrypted String enrypted="look it's unencrypted";
```

Der Weg zum eigenen Checker (Skizze)

Expert

<1> Definition einer Typ-Hierarchie

<2> Typ-Regeln: `BaseTypeVisitor` → Compiler Tree Api

<3> Typ-Inferenz: `@ImplicitFor` Annotation. Beispiel `RegexBottom`
`@ImplicitFor(literals = {LiteralKind.NULL}..)`

<4> ggf. DataFlow-Analyse: Subklassen von
`CFAbstractAnalysis` & `CFAbstractTransfer`

<5> Implementierung der Compiler Schnittstelle:
`SourceChecker` meist ← `BaseTypeChecker` oder `AggregateChecker`

Liste der mitgelieferten Checker

Aliasing Checker

Constant Value Checker

Fake Enum Checker

Format String Checker *

GUI Effect Checker

I18n Format String Checker

Interning Checker *

Linear Checker

Lock Checker

Map Key Checker

Nullness Checker *

Property File Checker

Reflection resolution Checker

Regex Checker *

Signature String Checker

Subtyping Checker *

Tainting Checker *

Units Checker *

Integration

IDEs & Buildsysteme

Checker Framework Integration

<> **Wrapper oder -processor**
- javacheck: Wrapper für javac
- javac -processor

<> **javacheck: Für Kommandozeilentools**

<> **-processor: mvn, gradle, ...**
- `-Xbootclasspath/p:${annotatedJdk}`
- `-processingpath checker.jar`
- Java 7: `-J-Xbootclasspath/p:${typeAnnotationsJavac}`
- Weitere Argumente, bspw. Warn modus: `-Awarns`

<> **Automatisches Processing:**

`META-INF/services/javax.annotation.processing.Processor`

Checker Framework Class Path

<> **Klassenpfad für**
- jdk{7,8}.jar: Boot Class Path
- checker-qual.jar: compile / compileOnly CP
- checker.jar: +processing path

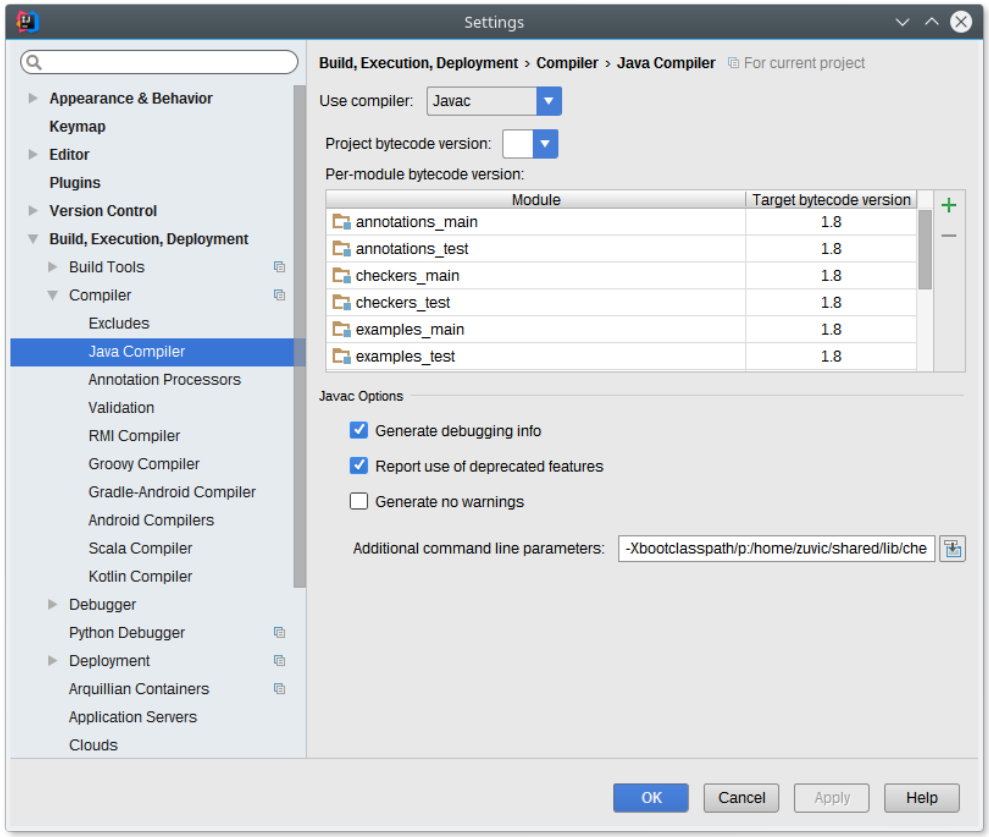
<> **Annotierte Fremdbibliotheken vs.** `-Astubs=`

<> **JDK 9?**

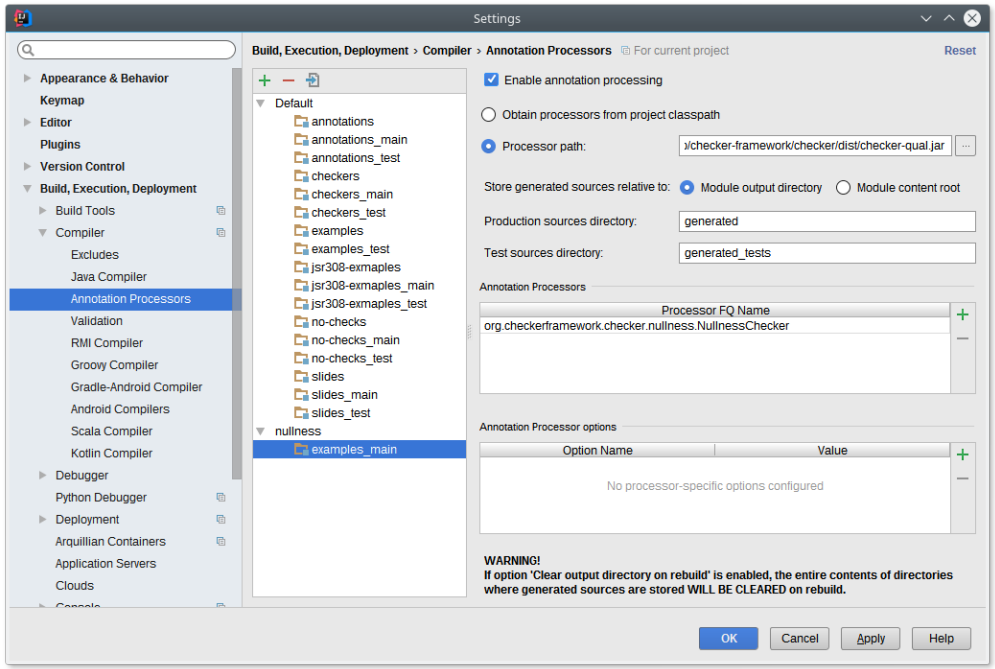
> **The Checkers Framework must continue to work.**
> —JEP 217: Annotations Pipeline 2.0

IDE Einrichten Beispiel Intellij

Expert



Processor in Settings → Build... → Compiler



Processor in Settings → Build... → Processing

Fazit

Nachteile & Probleme

<> **Eigene Typ-Qualifier → komplex**
- Compiler API
- Handbuch: Typ-Designer
- Test=Compilieren

<> **Compilierung etwas langsamer**

<> **Nicht eigängige Fehler**

<> **Integration: *Processing Classpath*?**

Vorteile

<> **Plug-In Verfikiation**
- Weiniger Tests
- Weniger `x!=null && x.doSomething()`
- Programmfluss bestimmen: SQL check
- Große Auswahl an Checker

<> **Flexibel (Eigene Typprüfung)**

<> **Keine Laufzeit Änderung**

<> **Typen können feiner Dokumentiert werden**

IMHO

<> **Sinnvolles, Mächtiges und Komplexes Werkzeug**

<> **Nicht vertieft:**

- *Typ-Polymorphie (Generics)*
- *Invariante Array Types*
- *Kontextsensitive Typ-Inferenz*
- *Type Refinement* (flow-sensitive)
- Java Expressions:@EnsuresNonNullIf, @EnsuresQualifierIf, @GuaredBy ...

<> **Dokumentation Typen → Besseren Java Code**

Tipps zum Einsteigen

Expert

<> **Tipps zum Einsteigen**

1. Nicht mit NullChecker beginnen
2. Sub-Typing als Type-Annotations
3. `infer-and-annotate.sh`
4. `-Awarns`
5. Sinnvoll ab Java 8
6. Online Demo Checker Framework

Referenzen

Checker Framework
http://checkerframework.org/
Online Demo Checker Framework
http://eisop.uwaterloo.ca/live (Source)
Data Flow Handbuch
http://types.cs.washington.edu/checker-framework/current/checker-framework-dataflow-manual.pdf
Pluggable Type Systems
http://bracha.org/pluggable-types.pdf
The Hitchhiker's Guide to javac
http://openjdk.java.net/groups/compiler/doc/hhgtjavac/index.html
JEP 217: Annotations Pipeline 2.0
http://openjdk.java.net/jeps/217

Fragen und Antworten

Danke...

Fragen und Antworten

Vortrag & Quellen

github.com/dzuvic/jsr308-examples

→

Dragan Zuvic
@dzuvic

www.w11k.de
www.thecodecampus.de