

Annotációk a Java programozási nyelvben

Jeszenszky Péter
Debreceni Egyetem, Informatikai Kar
jeszenszky.peter@inf.unideb.hu

Utolsó módosítás: 2018. december 19.

Történet (1)

- Az annotációk a 2004-ben kiadott J2SE 5-ben jelennek meg.
 - Lásd:
 - *New Features and Enhancements J2SE 5.0*
<https://docs.oracle.com/javase/1.5.0/docs/relnotes/features.html>
 - *JSR 175: A Metadata Facility for the Java Programming Language (Final Release)*. 30 September 2004. <https://jcp.org/en/jsr/detail?id=175>
- A 2006-ban megjelent Java SE 6 további lehetőségeket biztosít (`javax.annotation.processing` csomag).
 - *JSR 269: Pluggable Annotation Processing API (Final Release)*. 11 December 2006. <https://jcp.org/en/jsr/detail?id=269>

Annotáció fogalma

- Egy programkonstrukcióra vonatkozó metaadat, melynek nincs közvetlen hatása a programvégrehajtásra.

Történet (2)

- A 2014-ben megjelent Java SE 8 hoz újabb újdonságokat (típus annotációk, ismételhető annotációk, új előre definiált annotáció típusok).
 - Lásd: *What's New in JDK 8*
<https://www.oracle.com/technetwork/java/javase/8-whats-new-2157071.html>

Történet (3)

- Java SE 9:
 - *JEP 277: Enhanced Deprecation*
<http://openjdk.java.net/jeps/277>

5

Más nyelvek ekvivalens eszközei

- **.NET:** attribútumok
 - *.NET Framework Development Guide – Extending Metadata Using Attributes*
<https://docs.microsoft.com/en-us/dotnet/standard/attributes/index>
- **Python:** változó és függvény annotációk (a 3.0 verzió óta)
 - *PEP 526 – Syntax for Variable Annotations*
<https://www.python.org/dev/peps/pep-0526/>
 - *PEP 3107 – Function Annotations*
<https://www.python.org/dev/peps/pep-3107/>

7

Lehetséges felhasználások

- **Információk szolgáltatása a fordítónak:** Például tekintsen el bizonyos figyelmeztetésektől, jelezen bizonyos hibákat.
 - Lásd például a `@Deprecated` és `@Override` annotációkat.
 - *The Checker Framework* <https://checkerframework.org/>
- **Kódgenerálás:** Az annotációk alapján kód generálható.
 - Például a JAXB az annotációk révén XML dokumentumokat képes kezelni.
- **Futásidejű feldolgozás:** Bizonyos annotációkhoz hozzá lehet férni végrehajtási időben.
 - *JUnit* egységteszt keretrendszer <https://junit.org/>
 - *Bean Validation*: a Java EE része (lásd a `javax.validation` csomagot és alcsomagjait)
 - JSR 380: *Bean Validation 2.0 (Final Release)*. 3 August 2017. <https://jcp.org/en/jsr/detail?id=380>
 - Referencia implementáció: *Hibernate Validator* <http://hibernate.org/validator/>

6

Specifikáció

- James Gosling, Bill Joy, Guy Steele, Gilad Bracha, Alex Buckley, Daniel Smith. *The Java Language Specification – Java SE 11 Edition*. 21 August 2018.
<https://docs.oracle.com/javase/specs/jls/se11/html/>
- Lásd a következő részeket:
 - 9.6. *Annotation Types*
<https://docs.oracle.com/javase/specs/jls/se11/html/jls-9.html#jls-9.6>
 - 9.7. *Annotations*
<https://docs.oracle.com/javase/specs/jls/se11/html/jls-9.html#jls-9.7>

8

Annotációk szerkezete (1)

- A következők alkotják:
 - Egy annotáció típus neve.
 - Opcionálisan egy olyan lista, melyet vesszővel elválasztott elem-érték párok alkotnak.
 - A listát () karakterek között kell megadni.
- A névnek megfelelő annotáció típus határozza meg a használható elem-érték párokat.
 - Nem kötelező az alapértelmezett értékkel rendelkező elem-érték párok megadása.
- Az elem-érték párok sorrendje nem lényeges.
 - Az elem-érték párokat abban a sorrendben szokás egy annotációban megadni, melyben az annotáció típus deklarációjában is deklarálásra kerülnek az elemek.

9

Annotációk szerkezete (2)

- Ha egy elem típusa egy tömb típus, akkor az értéket egy tömb inicializáló kifejezés kell, hogy szolgáltatassa.
 - Kivéve azt az esetet, amikor az érték egy egyelemű tömb, ilyenkor elhagyható a kapcsos zárójelpár.
- Ekvivalens például az alábbi két annotáció:
 - `@Target({ElementType.METHOD})`
 - `@Target(ElementType.METHOD)`

11

Annotációk fajtái

- **Közönséges annotáció:**
 - `@XmlElement(name = "birthday", namespace = "http://xmlns.com/foaf/0.1/", required = true)`
- **Egyelemű annotáció:**
 - `@SuppressWarnings(value = "unchecked"), @SuppressWarnings("unchecked")`
 - `@Target(value = {ElementType.FIELD, ElementType.METHOD})`
`@Target({ElementType.FIELD, ElementType.METHOD})`
- **Jelölő annotáció:** ha nincs megadva egyetlen elem-érték pár sem, akkor elhagyhatók a () karakterek.
 - `@NotNull, @NotNull()`

10

Hol alkalmazható annotáció?

- Deklarációkra:
 - Annotáció típus, konstruktor, osztályváltozó, enum konstans, lokális változó, metódus, modul, csomag, formális paraméter, osztály, interfész és enum deklarációjára, típusparaméter deklarációjára (Java SE 8)
- Típusok használatára (Java SE 8)

12

Előre definiált annotáció típusok

- A `java.lang` csomagban:
 - `@Deprecated`
 - `@FunctionalInterface` (Java SE 8)
 - `@Override`
 - `@SafeVarargs` (Java SE 8)
 - `@SuppressWarnings`
- A `java.lang.annotation` csomagban:
 - `@Documented`
 - `@Inherited`
 - `@Native` (Java SE 8)
 - `@Repeatable` (Java SE 8)
 - `@Retention`
 - `@Target`

13

@Deprecated (2)

```
// Character.java (JDK 8):
package java.lang;

public final class Character implements java.io.Serializable,
    Comparable<Character> {
    ...

    /**
     * Determines if the specified character is permissible as the first
     * character in a Java identifier.
     * ...
     * @param ch the character to be tested.
     * @return {@code true} if the character may start a Java
     *         identifier; {@code false} otherwise.
     * ...
     * @deprecated Replaced by isJavaIdentifierStart(char).
     */
    @Deprecated
    public static boolean isJavaLetter(char ch) {
        return isJavaIdentifierStart(ch);
    }
    ...
}
```

15

@Deprecated (1)

- Az annotációval ellátott elem használata kerülendő, mert például veszélyes vagy jobb alternatíva létezik helyette.
 - Ajánlott a `@deprecated` Javadoc címkével is dokumentálni az annotált elem elavultságát.
- A fordítók figyelmeztetnek az annotációval ellátott elemek használatára.
- A Java SE 11 elavult elemei:
<https://docs.oracle.com/en/java/javase/11/docs/api/deprecated-list.html>

14

@Deprecated (3)

- A Java SE 9 bevezeti két opcionális elem használatát:
 - `since`: annak jelzésére szolgál, hogy az annotált elem melyik verzióban lett elavult (alapértelmezett érték: `""`)
 - `forRemoval`: annak jelzésére szolgál, hogy az annotált elem a jövőben eltávolításra kerül (alapértelmezett érték: `false`)
- Lásd:
<https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/lang/Deprecated.html>

16

@Deprecated (4)

- Példa:

```
// Runtime.java (JDK 11):
package java.lang;

...
public class Runtime {
    ...

    /**
     * Not implemented, does nothing.
     *
     * @deprecated
     * This method was intended to control instruction tracing.
     * It has been superseded by JVM-specific tracing mechanisms.
     * This method is subject to removal in a future version of Java SE.
     *
     * @param on ignored
     */
    @Deprecated(since="9", forRemoval=true)
    public void traceInstructions(boolean on) {}
    ...
}
```

17

@SuppressWarnings (1)

- Azt jelzi a fordító számára, hogy el kell tekinteni az annotált elemen (és a benne tartalmazott elemeknél) az adott figyelmeztetésektől.
- Lásd: <https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/lang/SuppressWarnings.html>

19

@Deprecated (5)

- Java SE 9:

- Elavult JDK API elemek használatának észlelésére szolgáló parancssori statikus kódelemző eszköz (jdepscan).
 - Példa:
 - jdepscan commons-io-2.6.jar
 - jdepscan lib/*.jar
 - Lásd: <https://docs.oracle.com/javase/9/tools/jdepscan.htm>
- Elavultnak jelölt típus importálása és elavultnak jelölt tag statikus importálása a fordításnál nem eredményez figyelmeztetést.
 - Lásd: JEP 211: *Elide Deprecation Warnings on Import Statements*
<http://openjdk.java.net/jeps/211>

18

@SuppressWarnings (2)

- Példa:

```
@SuppressWarnings("unchecked")
public ArrayList<String> getMusketeers() {
    ArrayList musketeers = new ArrayList();
    musketeers.add("D'Artagnan");
    musketeers.add("Athos");
    musketeers.add("Aramis");
    musketeers.add("Porthos");
    return musketeers;
}
```

```
import java.util.Date;
...
@SuppressWarnings("deprecation")
public static Date getDDay() {
    return new Date(1944 - 1900, 6 - 1, 6);
}
```

20

@Override (1)

- Azt jelzi, hogy a megjelölt metódus felülír egy olyan metódust, mely egy őss osztályban került deklarálásra.
- Nem kötelező megadni metódusok felülírásakor, de segít a hibák elkerülésében.
- Lásd:
<https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/lang/Override.html>

21

@FunctionalInterface (1)

- Annak jelzésére szolgál, hogy egy interfész funkcionális.
 - A funkcionális interfészeknek pontosan egy explicit módon deklarált absztrakt metódusa van.
- Lásd:
<https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/lang/FunctionalInterface.html>

23

@Override (2)

```
// Integer.java (JDK 11):
package java.lang;

public final class Integer extends Number implements
    Comparable<Integer> {
    ...

    /**
     * Returns a hash code for this {@code Integer}.
     *
     * @return a hash code value for this object, equal to the
     *         primitive {@code int} value represented by this
     *         {@code Integer} object.
     */
    @Override
    public int hashCode() {
        return Integer.hashCode(value);
    }
    ...
}
```

22

@FunctionalInterface (2)

- Példa:

```
// FileFilter.java (JDK 11):
package java.io;

@FunctionalInterface
public interface FileFilter {

    boolean accept(File pathname);

}
```

24

@SafeVarargs (1)

- Változó argumentumszámú függvényeknél jelentkező figyelmeztetésektől szabadít meg.
- Lásd:
<https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/lang/SafeVarargs.html>

25

@Native (1)

- Azt jelzi, hogy annotált osztályváltozó egy olyan konstanst definiál, mely natív kódból is hivatkozható.
 - Felhasználható például C++ header állományok előállításához.
- Lásd:
<https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/lang/annotation/Native.html>

27

@SafeVarargs (2)

- Példa:

```
// Collections.java (JDK 11):
package java.util;

public class Collections {
    ...
    @SafeVarargs
    public static <T> boolean addAll(Collection<? super T> c,
        T... elements) {
        boolean result = false;
        for (T element : elements)
            result |= c.add(element);
        return result;
    }
    ...
}
```

26

@Native (2)

- Példa:

```
// Integer.java (JDK 11):
package java.lang;

public final class Integer extends Number implements
    Comparable<Integer> {
    /**
     * A constant holding the minimum value an {@code int} can
     * have, -2<sup>31</sup>.
     */
    @Native public static final int MIN_VALUE = 0x80000000;
    ...
}
```

28

Meta-annotációk (1)

- Más annotáció típus deklarációkra alkalmazható annotációk, melyeket a `java.lang.annotation` csomag tartalmaz:
 - `@Documented`
 - `@Inherited`
 - `@Repeatable`
 - `@Retention`
 - `@Target`

29

Meta-annotációk (3)

- **@Repeatable:**
 - A Java SE 8-ban jelent meg, azt jelzi, hogy az annotáció akár többször is alkalmazható ugyanarra a deklarációra vagy típusra (lásd később).
 - Lásd: <https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/lang/annotation/Repeatable.html>
- **@Retention:**
 - Meghatározza az annotáció tárolásának módját, az alábbi lehetőségek választhatóak:
 - **RetentionPolicy.SOURCE:** a fordító figyelmen kívül hagyja az annotációt.
 - **RetentionPolicy.CLASS:** a fordító eltávolja az annotációt a bájtkódban, de az futásidőben nem elérhető.
 - **RetentionPolicy.RUNTIME:** a fordító eltávolja az annotációt a bájtkódban és az futásidőben is hozzáférhető.
 - Lásd: <https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/lang/annotation/Retention.html>

31

Meta-annotációk (2)

- **@Documented:**
 - Azt jelzi, hogy az adott annotáció használata meg kell, hogy jelenjen az API dokumentációban (alapértelmezésben az annotációk nem jelennek meg a Javadoc program által előállított dokumentációban).
 - Lásd: <https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/lang/annotation/Documented.html>
- **@Inherited:**
 - Azt jelzi, hogy az adott annotáció típus automatikusan öröklődik (alapértelmezésben nincs öröklés).
 - Lásd: <https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/lang/annotation/Inherited.html>

30

Meta-annotációk (4)

- **@Target:**
 - Meghatározza, hogy az annotáció mely elemekre használható, az alábbi lehetőségek állnak rendelkezésre:
 - Annotáció típus deklarációja (`ElementType.ANNOTATION_TYPE`)
 - Konstruktor deklaráció (`ElementType.CONSTRUCTOR`)
 - Osztályváltozó, enum konstans deklarációja (`ElementType.FIELD`)
 - Lokális változó deklarációja (`ElementType.LOCAL_VARIABLE`)
 - Metódus deklaráció (`ElementType.METHOD`)
 - Modul deklaráció (`ElementType.MODULE`)
 - Csomagdeklaráció (`ElementType.PACKAGE`)
 - Formális paraméter deklarációja (`ElementType.PARAMETER`)
 - Osztály, interfész vagy enum deklarációja (`ElementType.TYPE`)
 - Típusparaméter deklarációja (`ElementType.TYPE_PARAMETER`)
 - Típus használata (`ElementType.TYPE_USE`)
 - Lásd: <https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/lang/annotation/Target.html>

32

Annotáció típus deklaráció (1)

- Új annotáció típus létrehozása az alábbi annotáció típus deklarációval történik:
 - *módosítók @interface név { deklarációk }*
- A fenti deklaráció egy speciális interfészt határoz meg.
 - A közönséges interfészekre vonatkozó szabályok nem mindegyike vonatkozik az annotáció típus deklarációkra.
 - A közönséges interfészekkel ellentétben például nem lehet generikus és nem adható meg szülőinterfész sem.
 - Minden annotáció típus közvetlen szuper-interfésze a `java.lang.annotation.Annotation`, mely egy közönséges interfész.

33

Annotáció típus deklaráció (3)

- Az annotáció típus deklaráció törzsében elhelyezett metódus deklarációk mindegyike egy elemet deklarál.
 - A metódus deklarációkban nem megengedettek formális paraméterek, típusparaméterek és `throws` kulcsszó sem.
 - A visszatérési típus határozza meg az elem típusát, mely a következők valamelyike lehet:
 - Primitív típus
 - `String`
 - `Class/Class<T1, ..., Tn>`
 - enum típus
 - Annotáció típus
 - Olyan tömb, mely elemeinek típusa az előzőek valamelyike
 - Az elemekhez alapértelmezett érték adható a `default` kulcsszóval.
 - Egyelemű annotációknál a `value` nevet szokás az elemnek adni.

35

Annotáció típus deklaráció (2)

- A deklaráció törzsében az alábbi deklarációk megengedettek:
 - Osztálydeklaráció
 - Interfész deklaráció
 - Konstans deklaráció, mint például:
 - `int MIN = 0;`
 - `int MAX = 10;`
 - Speciális metódus deklaráció

34

Annotáció típus deklaráció és használata – 1. példa

```
// Evolving.java:
@Documented
public @interface Evolving {
}

// Experimental.java:
@Documented
public @interface Experimental {
}

// Stable.java:
@Documented
public @interface Stable {
}
```

```
// Foo.java:
public class Foo {

    @Experimental
    public void a() {
    }

    @Evolving
    public void b() {
    }

    @Stable
    public void c() {
    }

    public void d() {
    }

}
```

36

Annotáció típus deklarációja és használata – 2. példa

```
// Stability.java:
@Documented
public @interface Stability {
    public enum Status {
        EXPERIMENTAL,
        EVOLVING,
        STABLE
    }
    Status value();
}
```

37

Annotáció típus deklarációja és használata – 3. példa

- Az annotáció csak metódus és konstruktor deklarációhoz adható meg:

```
// Stability.java:
@Documented
@Target({ElementType.METHOD, ElementType.CONSTRUCTOR})
public @interface Stability {
    public enum Status {
        EXPERIMENTAL,
        EVOLVING,
        STABLE
    }
    Status value();
}
```

39

Annotáció típus deklarációja és használata – 2. példa (folytatás)

```
// Foo.java:
public class Foo {

    @Stability(Stability.Status.EXPERIMENTAL)
    public void a() {
    }

    @Stability(value=Stability.Status.EVOLVING)
    public void b() {
    }

    @Stability(Stability.Status.STABLE)
    public void c() {
    }

    public void d() {
    }

}
```

38

Annotáció típus deklarációja és használata – 4. példa

- A fordító eltárolja az annotációt a bájtkódban, mely hozzáférhető futásidőben:

```
// Stability.java:
@Documented
@Target({ElementType.METHOD, ElementType.CONSTRUCTOR})
@Retention(RetentionPolicy.RUNTIME)
public @interface Stability {
    public enum Status {
        EXPERIMENTAL,
        EVOLVING,
        STABLE
    }
    Status value();
}
```

40

Annotáció típus deklarációja és használata – 4. példa (folytatás)

- A Foo osztályban deklarált és @Stability annotációval ellátott metódusok:

```
Arrays.stream(Foo.class.getDeclaredMethods())
    .filter(method -> method.isAnnotationPresent(Stability.class))
    .forEach(method -> System.out.printf("%s is STABLE\n",
        method));
// public void Foo.b() is STABLE
// public void Foo.c() is STABLE
// public void Foo.a() is STABLE
```

41

Annotáció típus deklarációja és használata – 4. példa (folytatás)

```
// StabilityUtil.java:
public class StabilityUtil {

    public static Method[] getMethodsWithStability(Class c,
        Stability.Status status) {
        return Arrays.stream(c.getDeclaredMethods())
            .filter(method -> method.isAnnotationPresent(Stability.class)
                && method.getAnnotation(Stability.class).value() == status)
            .toArray(Method[]::new);
    }
    ...
}
```

```
for (Method method : getMethodsWithStability(Foo.class,
    Stability.Status.STABLE)) {
    System.out.printf("%s is STABLE\n", method);
}
// public void Foo.c() is STABLE
```

43

Annotáció típus deklarációja és használata – 4. példa (folytatás)

- A Foo osztályban deklarált és @Stability(Stability.Status.STABLE) annotációval ellátott metódusok:

```
Arrays.stream(Foo.class.getDeclaredMethods())
    .filter(method -> method.isAnnotationPresent(Stability.class)
        && method.getAnnotation(Stability.class).value() ==
            Stability.Status.STABLE)
    .forEach(method -> System.out.printf("%s is STABLE\n", method));
// public void Foo.c() is STABLE
```

42

Annotáció típus deklarációja és használata – 5. példa

```
// Todo.java:
@Documented
public @interface Todo {
    public enum Priority {
        LOW,
        NORMAL,
        HIGH;
    }
    Priority priority();
    String assignedTo() default "";
}
```

44

Annotáció típus deklarációja és használata – 5. példa (folytatás)

```
// Foo.java:
public class Foo {

    @Todo(priority = Todo.Priority.NORMAL)
    public void a() {
        // ...
    }

    public void b() {
        // ...
    }

    @Todo(priority = Todo.Priority.HIGH,
          assignedTo = "me")
    public void c() {
        // ...
    }
}
```

45

Ismételhető annotációk (1)

- Ugyanannak az annotáció típusnak a többszöri alkalmazása a programkód egy adott részéhez (Java SE 8).
 - Tartalmazó annotáció típus szükséges.

47

Annotáció típus deklarációja és használata – 6. példa

```
// Pattern.java:
@Documented
public @interface Pattern {
    String regex();
    int flags() default 0;
    String message();
}
```

```
// Pattern.java:
public class Kamion {

    @Pattern(message = "Érvénytelen forgalmi rendszám",
            regex = "^F[I-Z][A-Z]-\\d{3}$")
    String rendszám;
    ...
}
```

46

Ismételhető annotációk (2)

```
// Schedule.java:
@Documented
@Target(ElementType.METHOD)
@Repeatable(Schedules.class)
public @interface Schedule {
    String month() default "";
    String dayOfMonth() default "";
    int hour() default 12;
    int minute() default 0;
}
```

```
// Schedules.java:
@Documented
@Target(ElementType.METHOD)
public @interface Schedules {
    Schedule[] value();
}
```

48

Ismételhető annotációk (3)

```
// Foo.java:
public class Foo {

    @Schedule(dayOfMonth = "last", hour = 23, minute = 59)
    public periodicActivity1() {
        // ...
    }

    @Schedule(dayOfMonth = "first", hour = 8)
    @Schedule(dayOfMonth = "last", hour = 16)
    public periodicActivity2() {
        // ...
    }

    @Schedule(month = "Apr", dayOfMonth = "29")
    @Schedule(month = "Jun", dayOfMonth = "29")
    public periodicActivity3() {
        // ...
    }
}
```

49

Típus annotációk (2)

- Típus annotáció deklarálása és használata:

```
// NonNull.java:
@Documented
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.TYPE_USE)
public @interface NonNull {
}
```

51

Típus annotációk (1)

- Annotáció típus alkalmazása típusokra vagy azok részeire (Java SE 8).

50

Típus annotációk (3)

- Típus annotáció deklarálása és használata (folytatás):
 - `@NonNull` String s = getString();
 - String s = (`@NonNull` String) o;
 - `@NonNull` String processString(`@NonNull` String s) {
 ...
}
 - void processList(`@NonNull` List<`@NonNull` Object> list) {
 ...
}
 - <T> void processArray(`@NonNull` T[] arr) { ... }
 - <T> void processArray(`@NonNull` T `@NonNull` [] arr) {
 ...
}

52

Típus annotációk (4)

- Példa: *The Checker Framework* (licenc: GPLv2) <https://checkerframework.org/>
 - Ellenőrző (*checker*): olyan eszköz, mely bizonyos hibákra figyelmeztet vagy biztosítja, hogy az adott hiba ne forduljon elő.
 - Az ellenőrzés fordítási időben történik.
 - Az Eclipse IDE-ben és parancssorban is használható.
 - JDK 8 szükséges a használatához!

53

A javax.annotation.processing csomag (1)

- Lehetővé teszi annotációk fordítási időben történő feldolgozását.
 - A Java SE 6-ban jelent meg.
 - Lásd: *JSR 269: Pluggable Annotation Processing* <https://jcp.org/en/jsr/detail?id=269>
- A csomag `AbstractProcessor` osztálya szolgál az annotációk feldolgozására.
 - Lásd: `javax.annotation.processing.AbstractProcessor` <https://docs.oracle.com/en/java/javase/11/docs/api/java.compiler/javax/annotation/processing/AbstractProcessor.html>

55

Típus annotációk (5)

- Példa: *The Checker Framework* (folytatás):
 - Kód és parancssori használat:

```
List<@NonNull String> list = new ArrayList<String>();  
list.add(null);
```

```
$ javac -cp /path/to/checker.jar:/path/to/javac.jar \  
-Xbootclasspath/p:/path/to/jdk8.jar -processor \  
org.checkerframework.checker.nullness.NullnessChecker \  
Foo.java Bar.java  
Foo.java:8: error: [argument.type.incompatible] incompatible  
types in argument.  
    list.add(null);  
        ^  
    found   : null  
    required: @Initialized @NonNull String  
1 error
```

54

A javax.annotation.processing csomag (2)

- Példa annotációk feldolgozásra:

```
// StabilityProcessor.java:  
@SupportedAnnotationTypes("Stability")  
public class StabilityProcessor extends AbstractProcessor {  
  
    public SourceVersion getSupportedSourceVersion() {  
        return SourceVersion.latestSupported();  
    }  
  
    public boolean process(Set<? extends TypeElement> annotations,  
        RoundEnvironment roundEnv) {  
        for (Element element :  
            roundEnv.getElementsAnnotatedWith(Stability.class)) {  
            Stability stability = element.getAnnotation(Stability.class);  
            final String message = String.format("%s is %s", element,  
                stability.value());  
            processingEnv.getMessager().printMessage(Kind.NOTE, message);  
        }  
        return false;  
    }  
}
```

56

A javax.annotation.processing csomag (3)

- Példa annotációk feldolgozásra (folytatás):
 - Parancssori használat:

```
$ javac StabilityProcessor.java
$ javac -processor StabilityProcessor Foo.java
Note: a() is EXPERIMENTAL
Note: b() is EVOLVING
Note: c() is STABLE
```

További ajánlott olvasnivaló

- Joshua Bloch. *Effective Java*. Third Edition. Addison-Wesley Professional, 2017.
<http://www.informit.com/store/effective-java-9780134685991>