

Annotációk a Java programozási nyelvben

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

Utolsó módosítás: 2023. február 21.

Annotáció fogalma

- Egy programkonstrukcióra vonatkozó metaadat, melynek nincs közvetlen hatása a programvégrehajtásra.
 - Az annotációk gépi feldolgozásra alkalmasak, fordítási vagy futási időben érhetőek el.
 - Dokumentációs célokat is szolgálnak: nagyon tömör dokumentációs formának tekinthetők.

Történet (1)

- Az annotációk a 2004-ben kiadott J2SE 5.0-ban jelentek 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>

Történet (2)

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

Történet (3)

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

Történet (4)

- Java SE 11:
 - Annotációk alkalmazhatók lambda kifejezések formális paramétereire.
 - Lásd: *JEP 323: Local-Variable Syntax for Lambda Parameters* <https://openjdk.java.net/jeps/323>

Lehetséges felhasználások (1)

- **Információk szolgáltatása a fordítónak:** például tekintsen el bizonyos figyelmeztetésektől, jelezzen bizonyos hibákat.
 - Lásd például a `@Deprecated` és `@Override` annotációkat.
 - *The Checker Framework* <https://checkerframework.org/>
- **Kódgenerálás:** az annotációk alapján kód generálható.
 - *Jakarta XML Binding* (JAXB)
<https://eclipse-ee4j.github.io/jaxb-ri/>
 - *Project Lombok* <https://projectlombok.org/>

Lehetséges felhasználások (2)

- **Futásidejű feldolgozás:** bizonyos annotációkhoz hozzá lehet férni végrehajtási időben.
 - Egységtesztelés: *JUnit* <https://junit.org/>
 - Perzisztencia:
 - *Jakarta Persistence* (JPA) <https://jakarta.ee/specifications/persistence/>
 - *Jdbi* <https://jdbi.org/>
 - *Jackson* <https://github.com/FasterXML/jackson-databind>
 - Függőség befecskendezés:
 - *Google Guice* <https://github.com/google/guice>
 - *Spring Framework* <https://spring.io/projects/spring-framework>

Más nyelvek ekvivalens eszközei (1)

- **Kotlin:** annotációk <https://kotlinlang.org/docs/annotations.html>
- **Scala:** annotációk <https://docs.scala-lang.org/tour/annotations.html>
- **.NET:** attribútumok
 - *Extending Metadata Using Attributes*
<https://docs.microsoft.com/en-us/dotnet/standard/attributes/>
- **PHP:** attribútumok
<https://www.php.net/manual/en/language.attributes.php>
- **Python:** változó és függvény annotációk (a 3.0 verzió óta)
 - *PEP 526 – Syntax for Variable Annotations*
<https://peps.python.org/pep-0526/>
 - *PEP 3107 – Function Annotations* <https://peps.python.org/pep-3107/>

Más nyelvek ekvivalens eszközei (2)

- **Rust:** attribútumok

<https://doc.rust-lang.org/reference/attributes.html>

- **Swift:** attribútumok

<https://docs.swift.org/swift-book/documentation/the-swift-programming-language/attributes/>

- **JavaScript:** dekorátorok (javaslat)

<https://github.com/tc39/proposal-decorators>

- **TypeScript:** dekorátorok (kísérleti)

<https://www.typescriptlang.org/docs/handbook/decorators.html>

Specifikáció

- James Gosling, Bill Joy, Guy Steele, Gilad Bracha, Alex Buckley, Daniel Smith, Gavin Bierman. *The Java Language Specification – Java SE 17 Edition*. 9 August 2021.
<https://docs.oracle.com/javase/specs/jls/se17/html/>
- Lásd a következő részeket:
 - 9.6. *Annotation Interfaces*
<https://docs.oracle.com/javase/specs/jls/se17/html/jls-9.html#jls-9.6>
 - 9.7. *Annotations*
<https://docs.oracle.com/javase/specs/jls/se17/html/jls-9.html#jls-9.7>

Annotációk szintaxisa (1)

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

Annotációk fajtái

- **Közönséges annotáció:**

- `@XmlElement(name = "creator",
namespace = "http://purl.org/dc/terms/",
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()`

Annotációk szintaxisa (2)

- Ha egy elem típusa egy tömb típus, akkor az értéket egy tömb inicializáló kifejezés kell, hogy szolgáltassa.
 - 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)`

Hol alkalmazható annotáció?

- Deklarációkra:
 - Konstruktor, osztályváltozó, enum konstans, lokális változó, metódus, modul, csomag, formális paraméter, osztály, interfész (beleértve az annotáció interfészeket is), enum, típusparaméter, rekord és rekord komponens (Java SE 16) deklarációjára
 - **Deklaráció annotációknak** nevezzük a deklaráció kontextusban előforduló annotációkat.
- Deklarációkban és kifejezésekben használt típusokra (Java SE 8):
 - **Típus annotációknak** nevezzük a típus kontextusban előforduló annotációkat.

Annotációk jelentése

- Az előre definiált annotáció interfészeket kivéve a nyelvben nincs az annotációknak speciális jelentése.
- Azonban az előre definiált annotáció interfészek annotációinak speciális jelentése van, mely a fordításra is hatással van.

Előre definiált annotáció interfészek

- 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`

@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 9-től kezdve elavultnak jelölt típus importálása és elavultnak jelölt tag statikus importálása a fordításnál nem eredményez figyelmeztetést.
 - Lásd: *JEP 211: Elide Deprecation Warnings on Import Statements*
<https://openjdk.java.net/jeps/211>

@Deprecated (2)

- A Java SE 17 elavult elemei:
<https://docs.oracle.com/en/java/javase/17/docs/api/deprecated-list.html>

@Deprecated (3)

```
// Character.java (OpenJDK 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);
    }
    ...
}
```

@Deprecated (4)

- 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/17/docs/api/java.base/java/lang/Deprecated.html>

@Deprecated (5)

- Példa:

```
// Runtime.java (OpenJDK 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) {}
    ...
}
```

@Deprecated (6)

- `jdeprscan`:
 - Elavult JDK API elemek használatának észlelésére szolgáló parancssori statikus kódelemző eszköz.
 - Lásd:
<https://docs.oracle.com/en/java/javase/17/docs/specs/man/jdeprscan.html>
 - Használat:

```
$ jdeprscan commons-io-2.11.0.jar  
$ jdeprscan lib/*.jar
```

@SuppressWarnings (1)

- Azt jelzi a fordító számára, hogy el kell tekinteni az annotált elemen (és a benne tartalmazott programelemeknél) az adott figyelmeztetésektől.
- Lásd:
<https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/SuppressWarnings.html>

@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);
}
```

@Override (1)

- Azt jelzi, hogy a megjelölt metódus felülír egy olyan metódust, mely egy őssosztályban került deklarálásra.
- Nem kötelező megadni metódusok felülírásakor, de segít a hibák megelőzésében.
- Lásd:
<https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/Override.html>

@Override (2)

```
// Integer.java (OpenJDK 17):
package java.lang;

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

    /**
     * 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);
    }
    . . .
```

@FunctionalInterface (1)

- Annak jelzésére szolgál, hogy egy interfész funkcionális.
 - A funkcionális interfészeknek pontosan egy absztrakt metódusa van.

- Lásd:

<https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/FunctionalInterface.html>

@FunctionalInterface (2)

- Példa:

```
// FileFilter.java (OpenJDK 17):  
package java.io;  
  
@FunctionalInterface  
public interface FileFilter {  
    boolean accept(File pathname);  
}
```

@SafeVarargs (1)

- Változó argumentumszámú függvényeknél jelentkező figyelmeztetésektől szabadít meg.
- Lásd:
<https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/SafeVarargs.html>

@SafeVarargs (2)

- Példa:

```
// Collections.java (OpenJDK 17):  
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;  
    }  
    ...  
}
```

@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/17/docs/api/java.base/java/lang/annotation/Native.html>

@Native (2)

- Példa:

```
// Integer.java (OpenJDK 17):  
package java.lang;  
  
public final class Integer extends Number implements  
    Comparable<Integer>, Constable, ConstantDesc {  
  
    /**  
     * A constant holding the minimum value an {@code int} can  
     * have, -231.  
     */  
    @Native public static final int MIN_VALUE = 0x80000000;  
    . . .  
}
```

Meta-annotációk (1)

- Meta-annotációnak nevezünk egy annotáció interfész deklaráción megjelenő annotációt.
- A meta-annotációk metaadatokat szolgáltatnak az annotáció interfészekről.
- A `java.lang.annotation` csomag az alábbi meta-annotációkat biztosítja:
 - `@Documented`
 - `@Inherited`
 - `@Repeatable`
 - `@Retention`
 - `@Target`

Meta-annotációk (2)

- **@Documented:**

- Azt jelzi, hogy a megjelölt annotáció interfész annotációinak használata meg kell, hogy jelenjen az API dokumentációban (alapértelmezésben az annotációk nem jelennek meg a javadoc eszköz által előállított dokumentációban).
- Lásd:
<https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/annotation/Documented.html>

- **@Inherited:**

- Azt jelzi, hogy egy annotáció interfész automatikusan öröklődik (alapértelmezésben nincs öröklés).
- Lásd:
<https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/annotation/Inherited.html>

Meta-annotációk (3)

- **@Repeatable:**

- A Java SE 8-ban jelent meg, azt jelzi, hogy a megjelölt annotáció interfész annotációi akár többször is alkalmazhatók ugyanarra a deklarációra vagy típus használatra (lásd később).
- Lásd:
<https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/annotation/Repeatable.html>

- **@Retention:**

- Meghatározza a megjelölt annotáció interfész annotációihoz a tárolás módját, az alábbi lehetőségek választhatóak:
 - **RetentionPolicy.SOURCE:** a fordító figyelmen kívül hagyja az annotációkat.
 - **RetentionPolicy.CLASS:** a fordító eltárolja az annotációkat a bájtkódban, de azok futásidőben nem elérhetők.
 - **RetentionPolicy.RUNTIME:** a fordító eltárolja az annotációkat a bájtkódban és azok futásidőben is hozzáférhetők.
- Lásd:
<https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/annotation/Retention.html>

Meta-annotációk (4)

- **@Target:**

- Meghatározza, hogy az annotáció interfész mely elemekre használható, az alábbi lehetőségek állnak rendelkezésre:
 - Annotáció interfész 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`)
 - Rekord komponens deklaráció (`ElementType.RECORD_COMPONENT`)
 - Osztály, interfész (annotáció interfész is), enum vagy rekord deklarációja (`ElementType.TYPE`)
 - Típusparaméter deklarációja (`ElementType.TYPE_PARAMETER`)
 - Típus használata (`ElementType.TYPE_USE`)
- Lásd: <https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/annotation/Target.html>

Annotáció interfész deklarációja (1)

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

Annotáció interfész deklarációja (2)

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

Annotáció interfész 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.
 - Nem megengedettek a `private`, `default` és `static` módosítók.
 - A visszatérési típus határozza meg az elem típusát, mely a következők valamelyike lehet:
 - Primitív típus
 - `String`
 - `Class/Class< T_1, \dots, T_n >`
 - 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.

Annotáció interfész deklarációja 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() {
    }

}
```

Annotáció interfész deklarációja és használata – 2. példa

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

Annotáció interfész 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() {  
    }  
  
}
```

Annotáció interfész 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 {  
    enum Status {  
        EXPERIMENTAL,  
        EVOLVING,  
        STABLE  
    }  
    Status value();  
}
```

Annotáció interfész 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 {  
    enum Status {  
        EXPERIMENTAL,  
        EVOLVING,  
        STABLE  
    }  
    Status value();  
}
```

Annotáció interfész 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(System.out::println);  
// public void Foo.b()  
// public void Foo.c()  
// public void Foo.a()
```

Annotáció interfész 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.println(method + " is STABLE"));
// public void Foo.c() is STABLE
```

Annotáció interfész 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.println(method + " is STABLE");
}
// public void Foo.c() is STABLE
```


Annotáció interfész deklarációja és használata – 5. példa

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

Annotáció interfész 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() {
        // ...
    }

}
```

Annotáció interfész deklarációja és használata – 6. példa

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

```
// Book.java:  
public class Book {  
  
    @Pattern(regex = "^\\d{13}$", message = "Invalid ISBN number")  
    String isbn;  
    // ...  
  
}
```

Ismételhető annotációk (1)

- A Java SE 8-ban kerültek bevezetésre.
- Egy ismételhető annotáció interfész több annotációja is alkalmazható ugyanarra a programkonstrukcióra.
 - Fordítási hibát okoz, ha egy nem ismételhető annotáció interfész több annotációja is alkalmazásra kerül ugyanarra a programkonstrukcióra.
- Tartalmazó annotáció interfész szükséges.

Ismételhető annotációk (2)

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

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() {
        // ...
    }

}
```

Típus annotációk (1)

- Egy típus annotáció egy típusra (vagy annak egy részére) vonatkozó annotáció.
 - A Java SE 8-ban kerültek bevezetésre.
 - Egy típus annotáció annotáció interfészének deklarációja a `@Target (ElementType.TYPE_USE)` meta-annotációval megjelölt kell, hogy legyen.
- A típus annotációk hatékony eszközök hibák megelőzéséhez és érzékeléséhez, lásd például a *Checker Framework*-öt.

Típus annotációk (2)

- Típus annotáció deklarálása és használata:

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


Típus annotációk (3)

- Típus annotáció deklarációja 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) {
 ...
}`
 - `@NonNull var x = getData();`
 - `(@NonNull var x, @NonNull var y) -> x.process(y)`

Deklaráció annotációk és típus annotációk (1)

- A megfelelő annotáció interfész deklaráció határozza meg, hogy egy annotáció deklaráció annotáció vagy típus annotáció, vagy pedig mindkettő egyszerre.
 - Például az alábbi @Foo annotáció lehet deklaráció annotáció, típus annotáció, vagy mindkettő egyszerre:

```
@Foo private String s;
```

Deklaráció annotációk és típus annotációk (2)

- Az előbbi példában:
 - `@Foo` deklaráció annotáció s-en, ha `Foo` meta-annotálva van a `@Target(ElementType.FIELD)` annotációval.
 - `@Foo` típus annotáció `String`-en, ha `Foo` meta-annotálva van a `@Target(ElementType.TYPE_USE)` annotációval.

Deklaráció annotációk és típus annotációk (3)

- Lokális változó vagy lambda kifejezés formális paraméterének deklaráció annotációja soha nem kerül a `class` állományokban.
- Ezzel szemben lokális változó típusának vagy lambda kifejezés formális paraméter típusának annotációja tárolásra kerülhet a `class` állományokban.

Checker Framework (1)

- Lehetővé teszi a szoftverfejlesztők számára hibák észlelését és megelőzését a Java programjaikban.
 - Webhely: <https://checkerframework.org/>
 - Tároló:
<https://github.com/typetools/checker-framework>
 - Licenc: GPLv2

Checker Framework (2)

- Ellenőrző (*checker*): olyan eszköz, mely bizonyos hibákra figyelmeztet vagy biztosítja, hogy az adott hiba ne forduljon elő.
 - Az ellenőrzés fordítási időben történik.
- Használható az Eclipse és IntelliJ IDEA integrált fejlesztői környezetekben valamint a Gradle és Apache Maven fordítás-automatizáló rendszerekkel is.
- 8-as vagy későbbi verziójú JDK szükséges hozzá.

Checker Framework (3)

- Használat:

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

Checker Framework (4)

- Parancssori használat (JDK 17):

```
$ javac -J--add-exports=jdk.compiler/com.sun.tools.javac.code=ALL-UNNAMED \  
-J--add-opens=jdk.compiler/com.sun.tools.javac.comp=ALL-UNNAMED \  
-J--add-exports=jdk.compiler/com.sun.tools.javac.main=ALL-UNNAMED \  
-J--add-exports=jdk.compiler/com.sun.tools.javac.processing=ALL-UNNAMED \  
-J--add-exports=jdk.compiler/com.sun.tools.javac.tree=ALL-UNNAMED \  
-J--add-exports=jdk.compiler/com.sun.tools.javac.util=ALL-UNNAMED \  
-processorpath checker.jar \  
-cp checker-qual.jar \  
-processor org.checkerframework.checker.nullness.NullnessChecker \  
Foo.java Bar.java
```


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/17/docs/api/java.compiler/javax/annotation/processing/AbstractProcessor.html>

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;
    }
}
```

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
```

Eszközök és könyvtárak

- *Annotation File Utilities* (licenc: MIT License)
<https://checkerframework.org/annotation-file-utilities/>
<https://github.com/typetools/annotation-tools>
- *Annotations for JVM-based languages* (licenc: Apache License 2.0) <https://github.com/JetBrains/java-annotations>
- *Auto* (licenc: Apache License 2.0)
<https://github.com/google/auto>
- *Project Lombok* (licenc: MIT License)
<https://projectlombok.org/>
<https://github.com/projectlombok/lombok>

További ajánlott olvasnivaló

- *The Java Tutorials – Trail: Learning the Java Language – Lesson: Annotations.*
<https://docs.oracle.com/javase/tutorial/java/annotations/>
- Joshua Bloch. *Effective Java*. Third Edition. Addison-Wesley Professional, 2017.
<https://www.pearson.com/en-us/subject-catalog/p/effective-java/P2000000000138/>
 - Forráskód:
<https://github.com/jbloch/effective-java-3e-source-code>
- *Type Annotations FAQ*
<https://checkerframework.org/jsr308/jsr308-faq.html>