



Java 5 e Annotazioni

Alma Mater Studiorum - Università di Bologna
CdS Laurea Magistrale in Ingegneria Informatica
I Ciclo - A.A. 2017/2018
Sistemi Distribuiti M (8 cfu)

03 – Una Rapida Parentesi su Java 5 e Annotazioni (JSR-175)

Docente: Paolo Bellavista
paolo.bellavista@unibo.it

<http://lia.disi.unibo.it/Courses/sd1718-info/>
<http://lia.disi.unibo.it/Staff/PaoloBellavista/>



Java 5 e Annotation

- ❑ Velocissima panoramica su Java 5
- ❑ Il concetto di **annotazione** (annotation)
- ❑ **Tipi predefiniti di annotation**
- ❑ **Tipi personalizzati**: creazione e uso
- ❑ **Meta-annotation**
- ❑ Impostazione dei comportamenti di compilatore e JVM (*retention*)
- ❑ Ereditare annotation
- ❑ **Reflection & annotation**



J2EE5 e Ease of Development

Obiettivo cruciale di J2EE5 è **ease of development**

- ❑ Programmazione a componenti basata su **modello POJO** (Plain Old Java Objects), più vicina quindi al modello tradizionale di programmazione
 - Maggiore libertà, minori requisiti
- ❑ Utilizzo estensivo del concetto di **annotation**
 - **Necessità ridotte di descrittori di deployment**
- ❑ **Resource Injection**
 - **Inversione del controllo** (discuteremo del concetto ampiamente in una parte più avanzata del corso...)
- ❑ Alcune nuove API e funzionalità di supporto:
 - **JSR 220 (EJB 3.0)**
 - JSR 252 (JavaServer Faces 1.2), JSR 224 (JAX-WS 2.0 – Java API for XML Web Services), JSR 181 (WS Annotations), JSR 245 (JSP 2.1), JSR 222 (JAXB 2.0 – Java Architecture for XML Binding)
 - ...

Chi sa come funziona processo di specifica in JSR?



Brevissima Parentesi su J2EE6...

Per gli “addicted” alle evoluzioni tecnologiche e al versioning, in realtà siamo già giunti più che ampiamente ☺ a **Java EE v7** (Giugno 2013) – versione v8 rilasciata a Settembre 2017

Variazioni non particolarmente cruciali per programmazione enterprise distribuita; sicuramente meno centrali che non l’introduzione di J2EE5

- ❑ **Profili** come configurazioni differenziate della piattaforma Java EE
- ❑ **Annotation** specifiche per **servlet**
- ❑ Semplificazione modello e requisiti per **packaging applicazioni** Java EE
- ❑ Supporto a **RESTful Web Services** (JAX-RS)
- ❑ **Estensioni a dependency injection e context injection** (CDI – JSR 299)



I Metadati di Annotation

- ❑ **Elementi descrittivi (metadati)** associabili a:
 - package
 - classi e interfacce
 - costruttori
 - metodi
 - campi
 - parametri
 - variabili
- ❑ Informazioni che **aggiungono espressività agli elementi del linguaggio**
- ❑ Strutturati come **insiemi di coppie nome=valore**
- ❑ Lette e gestite dal compilatore o da strumenti esterni: **non influenzano direttamente la semantica** del codice ma il **modo in cui il codice può essere trattato da strumenti, VM e librerie**, che a sua volta può influenzare il comportamento runtime
- ❑ **Reperibili anche a runtime**



Motivazioni dell'Introduzione di Annotation in J2EE5

- ❑ **Molte API Java richiedono codice di integrazione** (“boilerplate code”). Idea di **generare automaticamente questo codice** se il codice di partenza è **“decorato” da opportuni metadati**. Ad esempio, accoppiamento interfaccia-implementazione in JAX-RPC
- ❑ **Arricchimento dello spazio concettuale** del linguaggio (verso **Declarative Programming**)
- ❑ Maggior potenza espressiva del linguaggio
- ❑ Possibilità di specificare informazioni relative a determinate entità (**code decoration**) **senza dover ricorrere a descrittori esterni** (ad es. evitando possibili disallineamenti fra file descrittori di deployment e codice corrispondente)



Motivazioni dell'Introduzione di Annotation in J2EE5

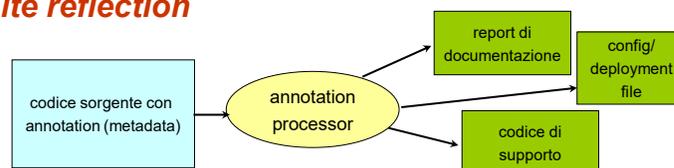
- Maggiore controllo di errori a compile-time

- **Semplicità di utilizzo**

Ma in realtà avete già utilizzato meccanismi simili: modificatore `transient`, `@deprecated` javadoc tag, ...

Ora **standardizzazione (JSR-175)** del modo con cui questi metadati possono essere aggiunti al codice

Importante: annotazioni possono (in quale caso?) essere lette da **file sorgenti, file class (bytecode) e runtime tramite reflection**



Ma partiamo bottom-up e cerchiamo prima di capire che cosa sono e come si usano...



Annotation Predefinite

- **Override:** per indicare che un determinato metodo **ridefinisce il corrispondente metodo ereditato da superclasse** (`java.lang.Override`)

```
@Override
public String toString() {
    return super.toString() + "[modified by subclass]";
}
```

Nota: nel caso di errore di trascrizione (del nome del metodo o dei parametri), con l'annotation `Override` il compilatore segnalerebbe l'errore (senza annotation verrebbe riconosciuto come un nuovo metodo della sottoclasse)

- **Deprecated:** indica che l'uso di un **metodo o di un tipo è sconsigliato** (`java.lang.Deprecated`)

```
@Deprecated
public class ExampleClass { ... }
```

- **SuppressWarnings:** **disattiva la segnalazione di warning** del compilatore per classi, metodi o campi (`java.lang.SuppressWarnings`)

```
@SuppressWarnings("unchecked")
public void aMethod() { ... }
```



Categorie di Annotation

- ❑ **Marker annotation**: *non hanno membri*; l'informazione è data dal nome stesso dell'annotazione, come nel caso di `Deprecated`
`@MarkerAnnotationName`
- ❑ **Single-value annotation**: *hanno un solo membro*, che deve chiamarsi *value*, come nel caso di `SuppressWarnings`
`@SingleValueAnnotationName("some value")`
- ❑ **Full annotation**: *annotation con più membri*
- ❑ **Anche tipi di annotation personalizzati**



Tipi di Annotation Personalizzati

- ❑ Dichiarazione di un tipo: **@interface**
- ❑ Dichiarazione di un metodo → elemento dell'annotazione
- ❑ Ogni tipo **estende automaticamente** l'interfaccia java.lang.annotation.Annotation

```
public @interface GroupTODO {
    public enum Severity {CRITICAL,IMPORTANT,TRIVIAL} ;

    Severity severity( ) default Severity.IMPORTANT;
    String item( );
    String assignedTo( );
}
```

UTILIZZO:

```
@GroupTODO (
    severity = GroupTODO.Severity.CRITICAL;
    item = "Figure out the amount of interest per month";
    assignedTo = "Paolo Bellavista";
)
public void calculateInterest(float amount , float rate) {
    //TODO }
}
```



Limitazioni

- ❑ **NON** si possono avere relazioni di **estensione** (**extends**) fra tipi di annotation
- ❑ I **tipi di ritorno** degli eventuali metodi di una annotation devono essere: tipi primitivi, String, Class, enum, tipi di annotation o array dei tipi elencati
- ❑ Una annotation **NON** può lanciare eccezioni, ovvero NON può avere una **throws clause**
- ❑ **NON** sono permessi **self-reference** (AnnotationA non può contenere un membro di tipo AnnotationA) né **circular-reference** (AnnotationA non può contenere un membro di tipo AnnotationB e questo di AnnotationA)



Riferimenti fra Annotation

```
public @interface Trademark
{
    String description( );
    String owner( );
}
```

```
public @interface License {
    String name( );
    String notice();
    boolean redistributable( );
    Trademark[ ] trademarks( );
}
```

```
@License (
    name = "SWIMM";
    notice= "license notice ...";
    redistributable = true;
    trademarks = {
        @Trademark(description = "abc" , owner = "xyz"),
        @Trademark(description = "efg" , owner = "klm")
    }
)
public class ExampleClass { ... }
```



Meta-Annotation: Annotazioni su Tipi Personalizzati di Annotation

- ❑ **@Target**: specifica il **tipo di elemento** al quale si può allegare tale tipo di annotation (field, method,...)

```
@Target ( { ElementType.METHOD,ElementType.PACKAGE } )  
public @interface ExampleAnnotation { ... }
```

- ❑ **@Documented**: specifica che le annotation di tale tipo faranno parte della **documentazione Javadoc** generata

```
@Documented  
public @interface ExampleAnnotation { ... }
```

- ❑ **@Inherited**: solo per annotazioni apposte a **classi**. Il tipo di annotation verrà automaticamente **ereditato dalle sottoclassi** della classe alla quale viene allegata

```
@Inherited  
public @interface ExampleAnnotation { ... }
```

- ❑ **@Retention**: politica di **mantenimento in memoria** con cui compilatore e JVM devono gestire le annotation



Politiche di Retention

- **@Retention(RetentionPolicy.SOURCE)**

Annotation permane solo a livello di codice sorgente → non memorizzata nel bytecode (.class file) → ignorata dalla JVM, utilizzata solo a tempo di sviluppo e compile-time, presente nel solo sorgente

- **@Retention(RetentionPolicy.CLASS) – default**

Annotation verrà registrata nel bytecode dal compilatore, ma non verrà mantenuta dalla JVM a runtime in modo ispezionabile da codice di «livello applicativo»; tipicamente utilizzabile a solo tempo di caricamento

- **@Retention(RetentionPolicy.RUNTIME)**

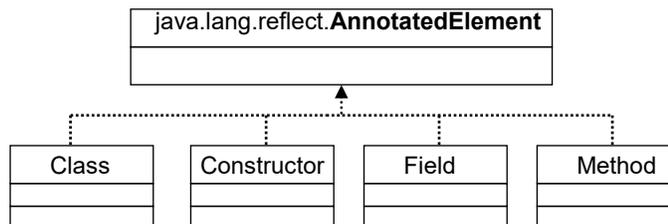
Annotation verrà registrata nel bytecode e potrà essere letta a runtime (mediante **reflection**) anche dopo il caricamento della classe da parte della JVM; utilizzabile anche all'interno del codice di supporto/applicativo a tempo di esecuzione, con proprietà eventualmente modificabili a runtime



Reflection & Annotation

Accesso runtime alle annotation (RetentionPolicy.RUNTIME):

→ introduzione interfaccia **AnnotatedElement**



- boolean **isAnnotationPresent**(Class<? extends Annotation> annotationType)
- <T extends Annotation>T **getAnnotation**(Class<T> annotationType)
- Annotation[] **getAnnotations**()
- Annotation[] **getDeclaredAnnotations**()



Esempio (1)

Verifica della presenza di una annotazione in una classe:

```
public void testAnnotationPresent(PrintStream out) throws
IOException{
    Class c = Sub.class;
    boolean inProgress = c.isAnnotationPresent(InProgress.class);
    if(inProgress){
        out.println("Sub is In Progress");
    } else {
        out.println("Sub is not In Progress");
    }
}
```

Output: "Sub is In Progress"



Esempio (2)

Verifica della presenza di una annotazione in una classe:

```
public void testGetAnnotation(PrintStream out) throws
    IOException, NoSuchMethodException{
    Class c = ExampleClass.class;
    AnnotatedElement el = c.getMethod("calculateInterest" ,
    "float.class", "float.class");
    GroupTODO groupToDo = el.getAnnotation(GroupTODO.class);
    String assignedTo = groupToDo.assignedTo();
    out.println("TODO item assigned to: " + assignedTo);
}
```

Output: "Paolo Bellavista"

Nota: Occorre aver associato all'annotation GroupTODO la meta-annotation `@Retention(RetentionPolicy.RUNTIME)`



Le Annotazioni, quindi...

- ❑ **Arricchimento dello spazio concettuale** del linguaggio (verso **Declarative Programming**)
- ❑ Maggior potenza espressiva del linguaggio
- ❑ Possibilità di specificare informazioni relative a determinate entità **senza dover ricorrere a descrittori esterni** (evitando possibili disallineamenti)
- ❑ Maggior controllo di errori a compile-time
- ❑ Semplicità di utilizzo

E ora ne vedremo immediatamente tutta una serie di **esemplificazioni e utilizzi pratici** nel modello dei componenti enterprise Java a partire da JEE5 (EJB v3.0)