UNIVERSITÀ PARTHENOPE
DiST
Dipartimento di Scienze e Tecnologie

Programmazione 3
e
Laboratorio di Programmazione 3

Tipi
Annotazioni

Angelo Ciaramella

Metadato

- Metadato
  - informazione sulle informazioni
  - vincoli (constraints)
- Esempio di metadato
  - Età rispetto alla definizione di persona
- Classe in Java
  - *metamodello* Java
- Annotazioni
  - meta-informazioni dirette verso un software

Annotazioni

- Annotazioni
  - tipo di struttura dati del linguaggio
  - si va ad aggiungere alle classi, alle interfacce e alle enumerazioni

```
public @interface DaCompletare {
String descrizione();
String assegnataA() default "da assegnare";
}
```

annotation elements

DaCompletare è un annotation type

equivalente alla dichiarazione di una variabile con relativo metodo omonimo che restituisce il suo valore

Nel caso lo sviluppatore non fornisca un valore

Annotazioni

```
public class Test {
public @DaCompletare(
descrizione = "Bisogna fare qualcosa...",
assegnataA = "Cristina"
)
public void faQualcosa() {
}
}
```

Il metodo `faQualcosa()` è stato annotato. Bisogna implementare un comportamento

Variabili invisibili

Annotazione ordinaria

- Annotazione completa (*full annotation*)

```
public @interface DaCompletare {
String descrizione();
String assegnataA() default "da assegnare";
}
```

Metodi astratti

```
public class DaCompletareImpl implements DaCompletare
{
private String descrizione;
private String assegnataA = "da assegnare";
public String descrizione() {
return descrizione;
}
public String assegnataA() {
return assegnataA;
}
}
```

Immaginando l'annotazione come un'interfaccia

Annotazione ordinaria

- All'interno di un'annotazione è possibile dichiarare
  - Metodi (implicitamente astratti)
  - Costanti
  - Enumerazioni
  - Tipi innestati

- Un'annotazione viene usata come se fosse un modificatore

```
@NomeAnnotazione ([lista di coppie] nome=valore)
```


Annotazione a valore unico

- Single value annotation
  - Contiene un singolo metodo `value()`

```
public @interface Serie {
Alfabeto value();
enum Alfabeto {A,B,C};
}
```

```
@Serie(value = Serie.Alfabeto.A)
public void faQualcosa() {
}
```

Esempio di utilizzo

Annotazione marcatrice

- Marker Annotation
  - Non ha metodi

```
public @interface Marker {}
```

```
@Marker()
public void faQualcosa() {
}
```

Esempio di utilizzo

Meta-annotazioni

- In `java.lang.annotation`
  - `Retention`
  - `Target`
  - `Documented`
  - `Inherited`
  - `Native (Java 8)`
  - `Repeatable (Java 8)`

- Servono ad annotare annotazioni

Target

- `java.lang.annotation.Target`
  - specifica gli elementi del linguaggio a cui è applicabile l'annotazione che si sta definendo

```
package java.lang.annotation;
public enum ElementType {
TYPE, // Classi, interfacce, o enumerazioni
FIELD, // variabili d'istanza (anche se enum)
METHOD, // Metodi
PARAMETER, // Parametri di metodi
CONSTRUCTOR, // Costruttori
LOCAL_VARIABLE, // Variabili locali o clausola catch
ANNOTATION_TYPE, // Tipi Annotazioni
PACKAGE // Package
TYPE_PARAMETER // Tipi parametro (visti con i
Generics)
TYPE_USE // Uso di un tipo
}
```

Enumerazione `ElementType`

Target

```
import java.lang.annotation.*;
import static java.lang.annotation.ElementType
@Target({TYPE, METHOD, CONSTRUCTOR, PACKAGE,
ANNOTATION_TYPE})
@Retention(RetentionPolicy.RUNTIME)
public @interface DaCompletare {
String descrizione();
String assegnataA() default "da assegnare";
}
```

Limitazione dell'annotazione `DaCompletare`

```
package java.lang.annotation;
@Documented
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.ANNOTATION_TYPE)
public @interface Target {
ElementType[] value( );
}
```

Definizione dell'annotazione `Target`

Type annotations

- Java 8
  - Uso dell'elemento dell'enumerazione `ElementType.TYPE_USE`

```
import java.lang.annotation.*;
@Target(ElementType.TYPE_USE)
@interface TestTP {
}
```

Esempio di annotazione

```
public class TypeParameterExample<@TestTP T> {
public void metodo (@TestTP T t){
System.out.println(t);
}
}
```

Esempio di utilizzo

Retention

- Retention
  - specifica come deve essere conservata dall'ambiente Java l'annotazione a cui viene applicata

```
package java.lang.annotation;
public enum RetentionPolicy {
SOURCE, // l'annotazione è eliminata dal compilatore
CLASS, /* l'annotazione viene conservata anche nel
file
".class", ma ignorata dall JVM */
RUNTIME /* l'annotazione viene conservata anche nel
file ".class", e letta dalla JVM */
}
```

Enumerazione `RetentionPolicy`

Retention

```
import java.lang.annotation.*;
@Retention(RetentionPolicy.RUNTIME)
public @interface DaCompletare {
String descrizione();
String assegnataA() default "da assegnare";
}
```

Le annotazioni di tipo `DaCompletare` sono conservate nei file compilati, per essere letti anche dalla JVM

Documented

- Documented
  - includere nella documentazione generata da Javadoc anche le annotazioni a cui è applicata

```
package java.lang.annotation;
@Documented
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.ANNOTATION_TYPE)
public @interface Target {
ElementType[] value( );
}
```

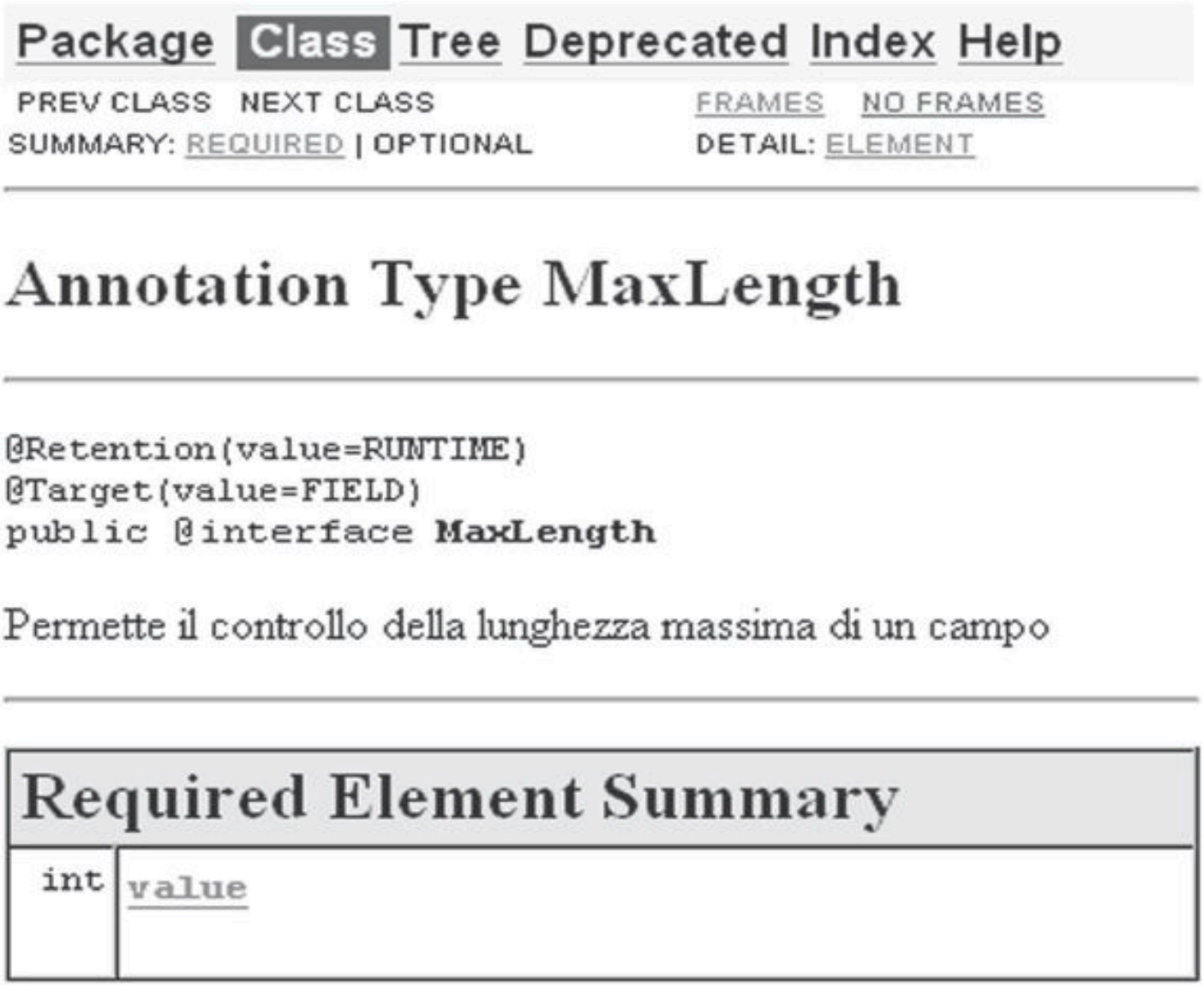
Meta-annotazione `Target`

Documented

```
import java.lang.annotation.*;
@Documented
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.FIELD)
public @interface MaxLength {
int value();
}
```

Lunghezza massima di un certo campo

Documented



Package **Class** Tree Deprecated Index Help

PREV CLASS NEXT CLASS FRAMES NO FRAMES

SUMMARY: REQUIRED | OPTIONAL DETAIL: ELEMENT

Annotation Type MaxLength

```
@Retention(value=RUNTIME)
@Target(value=FIELD)
public @interface MaxLength
```

Permette il controllo della lunghezza massima di un campo

Required Element Summary	
int	value

Documentazione dell'annotazione MaxLength

Inherited

- Inherited
  - le annotazioni applicate a classi possono essere ereditate

```
import java.lang.annotation.*;
import static java.lang.annotation.ElementType.*;
@Target({TYPE, METHOD, CONSTRUCTOR, PACKAGE,
ANNOTATION_TYPE})
@Retention(RetentionPolicy.RUNTIME)
@Inherited
public @interface DaCompletare {
String descrizione();
String assegnataA() default "da assegnare";
}
```

Annotazione

Inherited

```
@DaCompletare (
descrizione = "Da descrivere..."
)
public class SuperClasse {
//. . .
}
```

Annotazione applicata alla classe `SuperClasse`

```
public class SottoClasse extends SuperClasse {
//. . .
}
```

Estensione della classe `SuperClasse`

Repeatable

- Repeatable
  - annotare più volte con la stessa annotazione un elemento

```
@TestTrigger(quando="Ogni giorno, ore 18",
come = TestTrigger.StrumentoDiTest.JUNIT)
@TestTrigger(quando="Ogni venerdì, ore 9",
come = TestTrigger.StrumentoDiTest.GUI)
public class TestRepeatable {
public void metodo(String... args) {
// . . .
}
}
```

Annotazione `TestTrigger`

Annotazione standard

- `java.lang`
  - Override
  - Deprecated
  - SuppressWarning
  - Native
  - FunctionalInterface

Override

- Override
  - Annota un metodo con override

```
package java.lang;
import java.lang.annotation.*;
@Target(value=ElementType.METHOD)
@Retention(value=RetentionPolicy.SOURCE)
public @interface Override {}
```

Annotazione `java.lang.Override`

interpretabile solo dal compilatore e che non sarà inserita nel bytecode relativo

Il compilatore quindi potrà eventualmente segnalarci un problema nel caso il nome del metodo non sia scritto correttamente

Deprecated

- Deprecated
  - indica al compilatore e al runtime di Java che un metodo o un qualsiasi altro elemento di codice Java è deprecato

```
import java.lang.annotation.*;
import static
java.lang.annotation.RetentionPolicy.RUNTIME;
@Documented
@Retention(value=RUNTIME)
public @interface Deprecated {}
```

Annotazione `java.lang.Deprecated`

Deprecated

```
@DaCompletare (
descrizione = "Da descrivere ..."
)
public class SuperClasse {
/**
* Questo metodo è stato deprecato
* @deprecated utilizza un altro metodo per favore
*/
@Deprecated public void metodo() {
. . .
}
}
```

Esempio di utilizzo di `Deprecated`

SuppressWarnings

- SuppressWarnings
  - Se vogliamo che il compilatore non generi warnings

```
package java.lang;
import java.lang.annotation.*;
import java.lang.annotation.ElementType;
import static java.lang.annotation.ElementType.*;
@Target({TYPE, FIELD, METHOD, PARAMETER, CONSTRUCTOR,
LOCAL_VARIABLE})
@Retention(RetentionPolicy.SOURCE)
public @interface SuppressWarnings {
String[] value();
}
```

Annotazione SuppressWarnings

SuppressWarnings

```
@SuppressWarnings({"unchecked", "rawtypes"})
public static void main(String args[]) {
List strings = new ArrayList<String>();
strings.add("Lambda");
// . . .
Iterator<String> i = strings.iterator();
while (i.hasNext()) {
String string = i.next();
System.out.println(string);
}
}
```

Esempio di utilizzo di `SuppressWarnings`

FunctionalInterface

- `FunctionalInterface`
  - annotare una functional interface, ovvero una *interfaccia che ha un unico metodo astratto*

```
@Documented
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.TYPE)
public @interface FunctionalInterface {}
```

FunctionalInterface

Functional Interface

```
@FunctionalInterface
public interface FunctionalInterfaceExample {
void metodoAstratto();
void secondoMetodoAstratto();
default void metodoDiDefault(){
System.out.println("metodo di default");
}
}
```

Interfaccia con errore di compilazione

Native

- `Native`
  - permette di far interagire codice Java con codice scritto in linguaggio nativo, ovvero in C/C++

```
@Documented
@Target(ElementType.FIELD)
@Retention(RetentionPolicy.SOURCE)
public @interface Native {
}
```

Dichiarazione dell'annotazione `Native`

annotazione marcatrice applicabile solo alla dichiarazione di attributi ed in particolare alle costanti