Linux Day 2016 : Advanced JAVA

- Annotazioni e Reflection
- Espressioni Lambda

Annotazioni

Da Java 1.5

Annotazioni Comuni:

@Deprecated, @Override, e @SuppressWarnings

```
@interface Persona {
    String nome();
    String indirizzo() default "non assegnato";
   }
```

Annotazioni

USO:

- **Per informare il compilatore**
- **Per processare a tempo di compilazione (o di deploy)**
- **Per processare a run time (reflection)**

Uso pratico :

- **JPA (Hibernate, Eclipselink)**
- **Persistenza semplificata**

Annotazioni : Annotazione “Table”

```
@Retention(RetentionPolicy.RUNTIME) // esistente a runtime
@Target(ElementType.TYPE)                  //  a livello di classe
public @interface Table {
    String name();
    @Retention(RetentionPolicy.RUNTIME)
    public @interface Column {
        String name();
        String type() default "";  // possibile specificare il tipo
        int size() default -1;     // possibile specificare la lunghezza
    }
    @Retention(RetentionPolicy.RUNTIME)
    public @interface Id { }
    @Retention(RetentionPolicy.RUNTIME)
    public @interface Pwd {  }
}
```

Annotazioni: Classe annotata “Utente”

```
@Table(name = "utente")
public class Utente {
    private Integer id_utente;
    private String nome;                ...
@Table.Id
@Table.Column(name = "id_utente" )
      public Integer getId_utente() { return id_utente;  }
 @Table.Column(name = "nome" )
      public String getNome()        {  return nome;    }  ...
}
```

Reflection : Accesso ai dati

```
String getTableName(Class<?> classe) {
    return classe.getAnnotation(Table.class).name();
}

for (Method method : classe.getDeclaredMethods()) {
       if (method.isAnnotationPresent(Table.Column.class)) {
        elencoNomiCampi.add(
                method.getAnnotation(Table.Column.class).name());
         elencoTipiCampi.add(
                method.getReturnType().getSimpleName());
         elencoDatoCampi.add( method.invoke(utente));
           }
```

Lambda Expressions

- **Costituiscono una delle pricipali novità di Java 8**
- **Permettono di descrivere un metodo nel punto in cui viene utilizzato**
- **Hanno un tipo definito da un’interfaccia funzionale**

  (Essenzialmente un interfaccia con un solo metodo)

Lambda Expressions

```
interface Calcolo {
     int operaz(int a, int b);     }

public class Calcolatore {
public static int elabora(int a, int b,
Calcolo op) {
          return op.operaz(a, b);   }

public static void main(String[] args) {

    Calcolo molt= new Moltiplicazione();
    Calcolo somma = new Calcolo() {
         @Override
          public int operaz(int a, int b) {
                    return a + b; }
};
```

```
Calcolo sottrazione = (a, b) -> a - b;

Sys...println(elabora(2, 4, molt));
Sys...println(elabora(1, 2, somma));
Sys...println(elabora(3, 4, sottrazione));
Sys...println(elabora(3, 4,
                    (a,b)→Math.abs(a-b)));
                     // differenza assoluta
}   }

class Moltiplicazione implements Calcolo
{
        @Override
         public int operaz(int a, int b) {
         return a * b;     }
}
```

Lambda Expressions

Moltiplicazione :

```
class Moltiplicazione implements Calcolo {
        @Override
         public int operaz(int a, int b) {
         return a * b;     }
}

 Calcolo molt= new Moltiplicazione();

System.out.println(elabora(2, 4, molt));
```

Lambda Expressions

```
Calcolo somma = new Calcolo() {
        @Override
         public int operaz(int a, int b) {
                 return a + b; }
};

System.out.println(elabora(2, 4, somma));
```

Lambda Expressions

Sottrazione “Lambda”:

Calcolo sottrazione = (a, b) -> a – b;

System.out.println(elabora(2, 4, somma));

Differenza assoluta :

Sys...println(elabora(3, 4,(a,b)→Math.abs(a-b)));

Grazie