# **Example11**

#### Table of contents

- Aspect-Oriented Programming
  - 1. Shared Aspect
    - 1. <u>`it.unipr.informatica.aspects`</u>
      - 1. 'SharedAspect.java'
  - 2. Example11
    - 1. <u>`Example11.java`</u>

# **Aspect-Oriented Programming**

Oggetto remoto, astrarre il meccanismo di messaggi a qualcosa che possa scalare sulla rete. Utilizziamo i concetti della programmazione agli oggetti di JAVA 👙 per implementarlo.

Quando facciamo una richiesta al Cloud Amazon 3, che usa una lambda expression, vengono usati i concetti della programmazione agli aspetti.

L'obbiettivo: massimizzare il riuso del software.

Un *aspetto* lo intendiamo come aggettivo legato agli oggetti: voglio fare un oggetto che tutte le volte che cambia stato, questo viene salvato in un DB . Faremo attach() e detach() per attaccare e staccare l'aspetto.

- se ci riusciamo, possiamo attaccare e staccare gli aspetti a piacimento;
- se ci riusciamo, attacchiamo un oggetto a piu' aspetti;
- un insieme di aspetti, dovrebbe equivalere a un insieme di funzionalita' indipendenti dall'applicazione.

#### Tipi di aspetto:

#### condiviso

un oggetto condiviso puo' essere acceduto da tanti thread e deve garantire che il calcolo della risposta dei suoi messaggi venga realizzata in mutua esclusione; tutte le volte che entriamo in un metodo usiamo un lock, per esempio per accedere a una stringa;

### logging

per tracciare in modo permanente il comportamento del nostro sistema e identificare anomalie; prendere un oggetto e appiccicargli qualcosa che garantisca che tutti i messaggi/risposte siano tradotti;

### persistent

sopravvivono all'interruzione del nostro sistema, oggetti che devono mantenere il proprio stato; scegliamo noi quando salvare il nostro stato;

#### active

non chiedono in prestito il thread di esecuzione dal mittente del messaggio; eseguiamo i metodi in un thread pool a parte; remote

l'invocazione dei metodi da client remoti, ritorno del valori ed eccezioni ai client.

## **Shared Aspect**

Un *oggetto condiviso* e' uno i cui metodi vengono eseguiti in mutua esclusione, soltanto quelli delle interfacce implementate e non altro.

```
it.unipr.informatica.aspects
SharedAspect.java
```

Costruiamo una classe pubblica con metodo statico pubblico per permettere di attaccare l'oggetto. Sarebbe una classe factory e non vogliamo venga istanziata.

Vogliamo che l'utente chiami attach() passando un oggetto che chiama target di tipo T che ipoteticamente deve essere interfaccia. Il risultato sara' dello stesso tipo dell'argomento.

Con target.getClass() prendiamo la classe dell'oggetto target perche' vogliamo sapere tutte le interfacce che implementa; questo ci serve per creare il proxy. Facciamo cast sull'oggetto per assicurarci sia di tipo T e ritorniamo.

I messaggi non adranno finire all'oggetto target ma a proxy e siccome e' dinamico, quello che fa e' chiamare invoke() sull'invokation handler.

#### Lo stato:

- target passato come argomento;
- lock che definisce la sezione critica per guardia esecuzione condivisa;
- InnerInvocationHandler peristanziarsi.

```
this.target = target;
    this.lock = new Object();
}
//...
```

Prende il proxy a cui e' stato inviato il messaggio, il messaggio (nome esplicito), argomenti utilizzati.

Il try-catch serve ad attivare il lock.

```
// ...
                @Override
                public Object invoke
                (Object proxy, Method method, Object[] arguments)
                        throws Throwable {
                         try {
                                 synchronized (lock) {
                                         Object result = method.invoke(target,
arguments);
                                         return result;
                        } catch (InvocationTargetException exception) {
                                 throw exception.getCause();
                         } catch (Throwable throwable) {
                                 throw throwable;
                        }
                }
        }
}
```

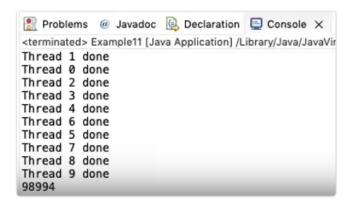
# Example11

## Example11.java

Costruiamo 10 thread: riempiono con 10k numeri interi una lista.

Ogni thread mette nella lista questi numeri, una volta che tutti sono terminati la lista conterra' 100k numeri interi.

Aspettiamo 5 secondi per evitare join() e altre funzioni.



Perche' non sono 100k gli elementi che ci aspettavamo?

Perche' LinkedList non sincronizza gli accessi, quando viene fatto add() da piu' thread, non abbiamo una sezione critica per far si' che non ci siano race condition. Puo' succedere che le add() non vadano a buon fine.

Sostituiamo la creazione della lista con il nostro shared aspect.

La mutua esclusione e' assicurata, ci impieghera' di piu' a rispondere ma avremo il nostro risultato.

```
List<Integer> list = SharedAspect.attach(new LinkedList<>());
```

