

Example12

Table of contents

- [Aspect-Oriented Programming](#)
 1. [Logging Aspect](#)
 2. [Example12](#)
 1. [`Example12.java`](#)
 3. [`it.unipr.informatica.aspects`](#)
 1. [`LoggingAspect.java`](#)

Aspect-Oriented Programming

Logging Aspect

Costruiamo un proxy, la chiamata invio di messaggio viene raccolta dal *proxy* che stampera' il **log** per eseguire la chiamata che verra' poi fatta.

Le informazioni essenziali che un record di log deve contenere sono:

1. il *thread di provenienza* della chiamata
in un sistema multi-thread, dobbiamo sapere quali chiamate vengono eseguite su uno e quali su un altro;
2. l'*ora del giorno*, ss/mm/hh, dd/mm/yyyy.

Se per qualche motivo viene passato un messaggio, senza attraversare il proxy, il log non verra' registrato. Dobbiamo quindi garantire che nessun messaggio venga passato senza passare prima attraverso il proxy: elimino il riferimento all'oggetto.

Questo l'abbiamo visto nell'esempio [Example11](#), quando facevamo

```
SharedAspect.attach(new LinkedList<>());
```

Example12

Example12.java

Un `TreeSet` e' un *albero binario* che organizza le sue foglie a profondita' minima (ampiezza massima) garantendo che queste vengano messe nel metodo classico, ovvero la radice del sinistro e' piu' piccola del destro.

`comparator` e' un oggetto che viene usato dal `TreeSet` tutte le volte che questo ha bisogno di confrontare 2 valori per decidere se proseguire sulla radice sinistra o destra.

≡ Esempio

`comparator` per `TreeSet` ordinamento di stringhe senza considerare maiuscole e minuscole: in questo modo 2 stringhe 'Ciao' e 'ciao' saranno le stesse.

Il `comparator` compara 2 oggetti dello stesso tipo.

```
public class Example12 {
    private void go() {
        Comparator<String> comparator =
            new CaseInsensitiveComparator();
        Set<String> set = new TreeSet<>(comparator);
        set.add("Verdi");
        set.add("Bianchi");
        set.add("Neri");
        set.add("Rossi");
        set.contains("Neri");
        set.contains("Viola");
    }

    public static void main(String[] args) {
        new Example12().go();
    }

    private static class CaseInsensitiveComparator
        implements Comparator<String> {
        @Override
        public int compare(String left, String right) {
            if (left == null || right == null)
                throw new IllegalArgumentException
                    ("o1 == null || o2 == null");
            return left.compareToIgnoreCase(right);
        }
    }
}
```

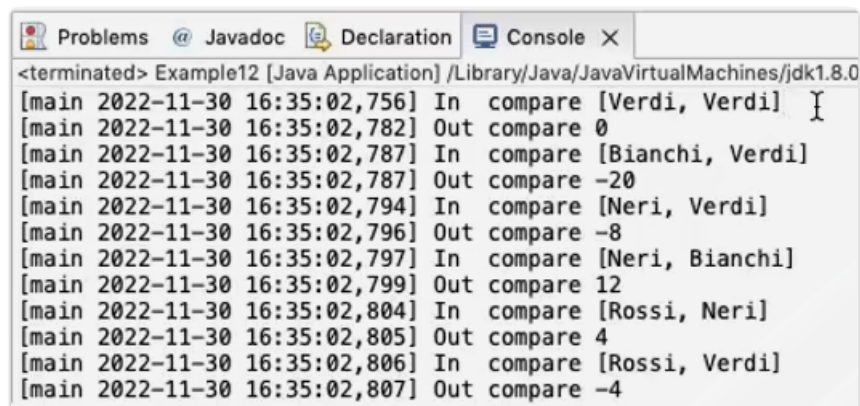
Il programma parte e non stampa niente.

Vogliamo garantire che tutte le volte che viene acceduto, ci dica il necessario.

Al nostro `comparator` sostituiamo il risultato dell'`attach()` del logging aspect:

```
comparator = LoggingAspect.attach(comparator);
```

Tutte le volte che il `TreeSet` chiama metodi su `comparator` (manda messaggi), li sta in verita' mandando al proxy che li intercetta, stampa e ritorna.



```
<terminated> Example12 [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0
[main 2022-11-30 16:35:02,756] In  compare [Verdi, Verdi]
[main 2022-11-30 16:35:02,782] Out  compare 0
[main 2022-11-30 16:35:02,787] In  compare [Bianchi, Verdi]
[main 2022-11-30 16:35:02,787] Out  compare -20
[main 2022-11-30 16:35:02,794] In  compare [Neri, Verdi]
[main 2022-11-30 16:35:02,796] Out  compare -8
[main 2022-11-30 16:35:02,797] In  compare [Neri, Bianchi]
[main 2022-11-30 16:35:02,799] Out  compare 12
[main 2022-11-30 16:35:02,804] In  compare [Rossi, Neri]
[main 2022-11-30 16:35:02,805] Out  compare 4
[main 2022-11-30 16:35:02,806] In  compare [Rossi, Verdi]
[main 2022-11-30 16:35:02,807] Out  compare -4
```

LoggingAspect.java

E' costruito esattamente come l'aspect che abbiamo visto in [Example11](#).

Costruiamo un proxy che usi lo stesso class loader del target, utilizzando le stesse interfacce, mettendoci un oggetto di classe `InvocationHandler`.

```
public class LoggingAspect {
    private static SimpleDateFormat DATE_FORMAT =
        new SimpleDateFormat("YYYY-MM-dd HH:mm:ss,SSS");

    public static <T> T attach(T target) {
        if (target == null)
            throw new IllegalArgumentException("target == null");
        Class<?> targetClass = target.getClass();
        Class<?>[] targetInterfaces = targetClass.getInterfaces();
        Object proxy =
            Proxy.newProxyInstance
                (targetClass.getClassLoader(), targetInterfaces,
                 new InnerInvocationHandler(target));
        @SuppressWarnings("unchecked")
        T result = (T) proxy;
        return result;
    }
    // ...
}
```

La `printf()` funziona nel modo identico alla sua parte originale in `C`.

- `%s` quale thread ha eseguito la chiamata;
- `%s` la stringa formattata di ora/giorno/mese;
- `%s` i dati passati al logger.

```
// ...
private static void log(String message) {
    String now = DATE_FORMAT.format(System.currentTimeMillis());
    System.out.printf
        ("%s [%s %s] %s\n",
         Thread.currentThread().getName(), now, message);
}

private static class InnerInvocationHandler
implements InvocationHandler {
    private Object target;
    private InnerInvocationHandler(Object target) {
        this.target = target;
    }
}
// ...
```

Compare e' l'unico metodo contenuto nell'interfaccia `Comparator`, viene stampato quello che abbiamo visto nell'immagine sopra.

Otteniamo un risultato oppure un'eccezione, in ogni caso scritte sempre in output.

```

// ...
@Override
public Object invoke
    (Object proxy, Method method, Object[] arguments)
    throws Throwable {
    String name = method.getName();
    try {
        log("In " + name + " " +
            Arrays.toString(arguments));
        Object result = method.invoke(target, arguments);
        log("Out " + name + " " + result);
        return result;
    } catch (InvocationTargetException exception) {
        Throwable cause = exception.getCause();
        log("Out " + name + " " + cause.getMessage());
        throw cause;
    } catch (Throwable throwable) {
        log("Out " + name + " " + throwable.getMessage());
        throw throwable;
    }
}
}
}

```