

Eccezioni

- Tanti linguaggi oggi hanno costrutti per definire eccezioni a livello di applicazioni, lanciare (throw) e catturare (catch) eccezioni
- Il codice di gestione delle eccezioni viene eseguito in una situazione anomala, ovvero fuori dalle condizioni di funzionamento normale del programma
- Problema
 - I programmi mischiano il codice che implementa il comportamento normale (quello che si dovrebbe fare) con il codice per rilevare e gestire una eccezione

E. Tramontana - Refactoring - Mar 2008 1

Esempio Store in Java

```
public class Store {
    private static String nomeFile = "Log_trans.txt";
    private static FileWriter f;
    public static void init() {
        try {
            f = new FileWriter(nomeFile);
        } catch (IOException e) {
            System.out.println("Apertura file non riuscita.");
        }
    }
    public static void write(String s) {
        Calendar c = Calendar.getInstance();
        s = "time "+c.getTime()+" "+s;
        try {
            f.write(s+" \n");
        } catch (IOException e) {
            System.out.println("Scrittura su disco non riuscita.");
        }
    }
}
```

E. Tramontana - Refactoring - Mar 2008 2

Gestione eccezioni

- In molti casi i codici che implementano “cosa fare” e come “rilevare e gestire una eccezione” sono mischiati (o addirittura nella stessa operazione)
- Problemi
 - Riutilizzo: non è possibile ridefinire solo cosa fare usando una sottoclasse, lo stesso per la gestione delle eccezioni
 - Evoluzione: quello che fa un metodo non può essere usato in un contesto con differenti eccezioni, senza cambiare il metodo
 - Semplicità: il codice diventa più difficile da comprendere
 - Mancanza di astrazione: l'implementazione della gestione di una situazione di fault (eccezione) è sparsa in varie parti del codice, rendendola difficile da esprimere e mantenere

E. Tramontana - Refactoring - Mar 2008 3

Soluzione ad Aspetti

- Tramite aspetti
 - Separo il codice che implementa il comportamento normale con il codice di gestione eccezioni
 - Elimino codice di gestione eccezioni sparso in varie classi
 - Elimino codice duplicato
 - Il codice di gestione delle eccezioni è riutilizzabile
 - Posso inserire e togliere gli aspetti (gestione eccezioni) quando voglio
 - Il codice delle classi risulta più semplice, più riutilizzabile
 - Sviluppo incrementale del software

E. Tramontana - Refactoring - Mar 2008 4

Exception Softening

- Java specifica due categorie di eccezioni: checked e unchecked
- Quando le eccezioni sono checked il codice chiamante deve gestirle (con try catch) oppure dichiarare che può lanciarle (con throw)
- Le eccezioni unchecked, RuntimeException o Error, non necessitano di essere gestite esplicitamente dal codice chiamante
- Exception softening permette di gestire le eccezioni checked come quelle unchecked
 - Questo permette di rendere il codice di gestione delle eccezioni più modulare

Formato:

```
declare soft : <Exception> : <pointcut>;
```

L'aspetto eseguirà il join point in un blocco try catch

E. Tramontana - Refactoring - Mar 2008 5

Esempio 2

```
import java.rmi.RemoteException;
```

```
public class TestSoftening {
    public static void main(String[] args) {
        TestSoftening test = new TestSoftening();
        test.perform();
    }

    public void perform() throws RemoteException {
        throw new RemoteException();
    }
}
```

E. Tramontana - Refactoring - Mar 2008 7

Esempio Store in AspectJ

```
public class Store {
    private static String nomeFile = "Log_trans.txt";
    private static FileWriter f;
    public static void init() {
        f = new FileWriter(nomeFile);
    }
    public static void write(String s) {
        Calendar c = Calendar.getInstance();
        s = "time "+c.getTime()+" "+s;
        f.write(s+" \n");
    }
}
```

```
public aspect SofteningAspect {
    declare soft : IOException : execution(void Store.*(..));
}
```

E. Tramontana - Refactoring - Mar 2008 6

Aspetto per gestione eccezione

```
import java.rmi.RemoteException;
```

```
public aspect SofteningAspect {
    declare soft : RemoteException : call(void TestSoftening.perform());

    pointcut handleExc() : call(void TestSoftening.perform());

    void around() : handleExc() {
        try {
            proceed();
        } catch (Exception e) {
            System.out.println("eccezione catturata dall'aspetto");
        }
    }
}
```

E. Tramontana - Refactoring - Mar 2008 8