

Participant Pattern

- Un aspetto cattura operazioni sulla base della signature dei metodi delle classi
- Si vuole poter catturare operazioni che hanno caratteristiche comuni (ma non lo stesso nome o nomi simili)
 - Es. tutte le operazioni che necessitano un lungo tempo di esecuzione, oppure
 - Tutte le operazioni che svolgono attività di IO, oppure
 - Le operazioni che servono a portare a termine una transazione
- Le operazioni da catturare sono definite da signature completamente differenti
 - Come definire i pointcut?

Ing. E. Tramontana - Participant, Lock - 14-Dic-06 1

Participant Pattern - Soluzione 1

- Definire un pointcut che elenca tutti i metodi che hanno la caratteristica comune (es. lungo tempo di esecuzione)

```
public aspect CommonAspect {  
    pointcut longexec() : call(* Account.checkBalance()) ||  
        call(* Bank.closeYear()) ||  
        call(* ATM.dumpData());  
  
    void around() : longexec() {  
        ...  
    }  
}
```

- Conseguenze
 - Bisogna modificare il pointcut per far avvenire la cattura quando
 - Si aggiungono nuovi metodi con la stessa caratteristica
 - Si cambiano i nomi di metodi esistenti
 - Si rende un metodo privo della caratteristica
 - Ovvero l'aspetto e le classi devono essere entrambi cambiati per rimanere coordinati

Ing. E. Tramontana - Participant, Lock - 14-Dic-06 2

Participant Pattern - Soluzione 2

- La caratteristica su cui l'aspetto agisce dipende da come l'operazione è implementata
 - È bene quindi definire il pointcut nella classe che implementa l'operazione
 - Nota: le classi possono definire i pointcut (ma non gli advice)
 - La classe partecipa alla definizione dell'aspetto

```
public class Account {  
    pointcut longexec() : call(* Account.checkBalance());  
    ...  
}  
  
public class Bank {  
    pointcut longexec() : call(* Bank.closeYear());  
    ...  
}  
  
public aspect CommonAspect {  
    void around() : Account.longexec() || Bank.longexec() {  
        ...  
    }  
}
```

Ing. E. Tramontana - Participant, Lock - 14-Dic-06 3

Participant Pattern - Soluzione 3

- Definire una annotazione per la caratteristica da catturare
- Usare l'annotazione definita per i metodi che hanno tale caratteristica

```
public @interface MarkLongExec { }  
  
public class Account {  
    @MarkLongExec  
    public void checkBalance() { ... }  
    ...  
}  
  
public class Bank {  
    @MarkLongExec  
    public void closeYear() { ... }  
    ...  
}
```

- Definire un aspetto che cattura l'annotazione

Ing. E. Tramontana - Participant, Lock - 14-Dic-06 4

Cattura metodi annotati

```
public aspect CommonAspect {  
  
    pointcut longexec() :  
        // cattura l'esecuzione dei metodi annotati  
        execution(@MarkLongExec * *.*(..));  
  
    void around() : longexec() {  
        System.out.println("in around");  
        proceed();  
    }  
}
```

Ing. E. Tramontana - Participant, Lock - 14-Dic-06 5

Considerazioni

- La soluzione 3 è quella più interessante
 - Poiché modulare
 - Lascia la definizione del pointcut separata dalle classi, cioè nell'aspetto
 - Poiché indipendente dalle classi
 - L'aspetto non fa assunzioni sulla signature delle operazioni
 - Ok, ... occorre annotare appositamente le operazioni (nel codice delle classi)

Ing. E. Tramontana - Participant, Lock - 14-Dic-06 6

Read-write lock pattern

- In un ambiente multi-thread, per massimizzare il parallelismo
 - Più lettori devono poter accedere contemporaneamente ad un dato
 - Un solo scrittore può modificare il dato
- Si usa una coppia di lock
 - Più lettori possono acquisire il lock in lettura, se non vi è nessuno scrittore
 - Un solo scrittore può acquisire il lock in scrittura se non vi è nessuno scrittore e nessun lettore

Ing. E. Tramontana - Participant, Lock - 14-Dic-06 7

Classe che usa i lock

```
import java.util.concurrent.*;  
public class Account {  
    ...  
    private ReadWriteLock rwlock = new ReentrantReadWriteLock();  
    public void credit(int amount) {  
        try {  
            rwlock.writeLock().lock();  
            setBalance(getBalance() + amount);  
        } catch (InterruptedException ex) { ... }  
        finally {  
            rwlock.writeLock().unlock();  
        }  
    }  
    ...  
}
```

Ing. E. Tramontana - Participant, Lock - 14-Dic-06 8

Soluzione OO

- Nella soluzione OO
 - Le operazioni `credit()` e `debit()` implementano la loro funzionalità principale ed alcune attività di sincronizzazione
 - Il concern di sincronizzazione è crosscutting ed il codice corrispondente è invasivo
 - Se si dimentica di gestire la sincronizzazione in uno dei metodi di `Account`, il sistema va potenzialmente in errore
 - Occorre fare attenzione ad acquisire il giusto tipo di lock (read o write) per ciascuna operazione

Ing. E. Tramontana - Participant, Lock - 14-Dic-06 9

Soluzione AO

- Nella soluzione AO
 - Si evita di modificare la classe `Account` per eseguire l'acquisizione dei lock
 - Un aspetto cattura le operazioni di tipo read e di tipo write ed acquisisce i corrispondenti lock

Ing. E. Tramontana - Participant, Lock - 14-Dic-06 10

Aspetto abstract per sincronizzazione

```
public abstract aspect ReadWriteLockSynchronizationAspect
    perthis(readOperations() || writeOperations()) {

    public abstract pointcut readOperations();
    public abstract pointcut writeOperations();
    private ReadWriteLock rwLock= new ReentrantReadWriteLock();
    before() : readOperations() {
        rwLock.readLock().lock();
    }
    after() : readOperations() {
        rwLock.readLock().unlock();
    }
    ...
}
```

Ing. E. Tramontana - Participant, Lock - 14-Dic-06 11

Aspetti e istanze

- In genere, un aspetto è istanziato automaticamente
 - Una singola istanza di un aspetto esiste per ciascun programma in esecuzione
 - Ovvero, la medesima istanza di un aspetto interagisce con tutte le istanze di tutte le classi del programma
 - Ovvero, ciascun aspetto è un Singleton
- È possibile associare un aspetto ad una singola istanza di una classe
 - `aspect nome perthis(pointcut) { }`Oppure
 - `aspect nome pertarget(pointcut) { }`

Ing. E. Tramontana - Participant, Lock - 14-Dic-06 12

Aspetto per sincronizzazione

```
public aspect BankingSynchronizationAspect extends
    ReadWriteLockSynchronizationAspect {
    public pointcut readOperations():
        execution(* Account.get*(..)) ||
        execution(* Account.toString(..));

    public pointcut writeOperations():
        execution(* Account.*(..)) &&
        !readOperations();
}
```