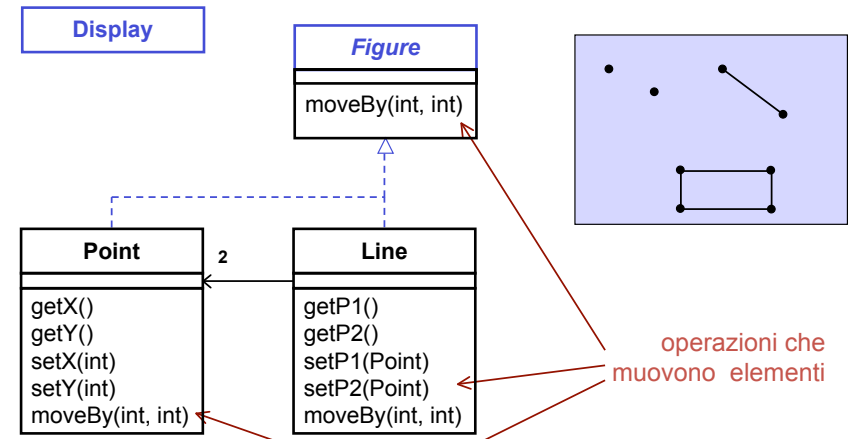


AOP

- Un aspetto è
 - Una entità (una sorta di classe) che permette di esprimere uno o più concern trasversali in modo modulare
 - Un costrutto di programmazione
- Esempi tipici di aspetti (e di concern trasversali)
 - Logging
 - Sincronizzazione
 - Sicurezza
 - Gestione memoria

Esempio: editor per figure



Note: the Editor example (shown in this slide and some of the following slides) is based on a presentation given by Kiczales

Codice editor per figure

```
class Line implements Figure {
    private Point p1, p2;
    Point getP1() { return p1; }
    Point getP2() { return p2; }
    void setP1(Point p1) { this.p1 = p1; }
    void setP2(Point p2) { this.p2 = p2; }
    void moveBy(int dx, int dy) { ... }
}

class Point implements Figure {
    private int x = 0, y = 0;
    int getX() { return x; }
    int getY() { return y; }
    void setX(int x) { this.x = x; }
    void setY(int y) { this.y = y; }
    void moveBy(int dx, int dy) { ... }
}
```

- Gli elementi che vengono spostati devono essere ridisegnati sul display

Crosscutting concern x display

- Senza aspetti, il codice di setP1(), setP2(), moveBy() contiene l'invocazione ad update() di Display

```
class Line {
    private Point p1, p2;

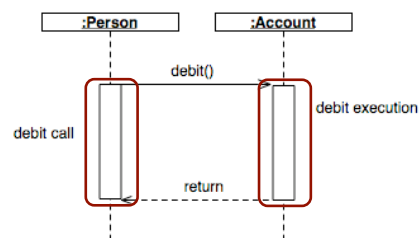
    Point getP1() { return p1; }
    Point getP2() { return p2; }

    void setP1(Point p1) {
        this.p1 = p1;
        Display.update();
    }

    void setP2(Point p2) {
        this.p2 = p2;
        Display.update();
    }
}
```

Modello dei join point

- Join point di metodi
 - `execution()` (lato esecuzione)
 - `call()` (lato chiamata)
- Join point di costruttori
 - `execution()`
 - `call()`
- Join point di accesso ai campi
 - Read `get()`
 - Write `set()`
- Join point di esecuzione gestione eccezioni
 - Associati al blocco `catch` di una eccezione
- Join point di inizializzazione classe
 - Intervengono quando una classe è caricata
- Join point di esecuzione advice



Aspetti

- Aspetti
 - Possono includere
 - campi e metodi (come le classi)
 - costruttori, ma senza argomenti
 - Possono specificare la loro visibilità (`public`, etc.)
 - Possono essere `abstract`
 - Per essere riusati da altri aspetti
 - Hanno dei metodi o dei `pointcut abstract`
 - Non intercettano niente (poiché `abstract`)
 - Possono estendere classi e aspetti `abstract` e `implem.` interfacce
 - Non possono essere istanziati direttamente (non si usa il `new` per un aspetto)
 - Tuttavia una unica istanza viene creata automaticamente
 - Non possono ereditare da aspetti non `abstract`

Dichiarazione di pointcut

- I pointcut
 - Identificano dei join point nel flusso del programma
 - Possono essere dichiarati per assegnare loro un nome
- Formato di un pointcut

```
[access] pointcut nome([param]) : designator(joinpoint);
```

- nome definisce un nome (stringa) per il pointcut
- designator indica quando si avrà l'associazione pointcut e joinpoint (es. `call`, `execution`, `get`, `set`)
- joinpoint è definito ad es. dalla signature di un metodo

Es. di pointcut

- Per catturare ciascuna chiamata al metodo che ha signature `void Point.setX(int)`
 - Il pointcut è `pointcut cattura() : call(void Point.setX(int));`
- Per l'interfaccia sarebbe `call(void Figure.setXY(int, int))`
 - Cattura ciascuna chiamata a `setXY(int, int)` nelle classi che implementano `Figure`
- Per catturare l'istanziamento di `Point` `call(public Point.new())`

Composizione di pointcut

- I pointcut possono usare operatori boolean &&, ||, !
- Per identificare e catturare entrambi i metodi setX() e setY() di Point
`call(void Point.setX(int)) || call(void Point.setY(int))`
- Per catturare metodi e tipi diversi
`call(void Point.setX(int)) ||
call(void Point.setY(int)) ||
call(void Line.setP1(Point))`
- Per catturare invocazioni a setX() ma al di fuori di DisplayUpdating
`call(void Point.setX(int)) && !within(DisplayUpdating)`
- Per catturare tutti i join point dentro un modulo (es. classe) mentre che il codice esegue: within()
Ing. E. Tramontana - AOP - 8-Nov-06 9

Aspetto per editor di figure - 2

- Aspetto per catturare le chiamate a metodi di più classi

```
aspect DisplayUpdating {  
    pointcut move():  
        call(void Figure.moveBy(int, int)) ||  
        call(void Line.setP1(Point)) ||  
        call(void Line.setP2(Point)) ||  
        call(void Point.setX(int)) ||  
        call(void Point.setY(int));  
  
    after() returning: move() {  
        Display.update();  
    }  
}
```

Aspetto per editor di figure

- Con aspetti
 - Definiamo un pointcut che cattura setP1(), setP2(), moveBy() e un advice che invoca update() di Display
 - Es. `call(void Line.setP1(Point))`
 - Le invocazioni di update() non sono sparse tra i vari metodi (e le varie classi Line, Point)

```
aspect DisplayUpdating {  
    pointcut move():  
        call(void Line.setP1(Point)) ||  
        call(void Line.setP2(Point));  
  
    after() returning: move() {  
        Display.update();  
    }  
}
```

- Aspetto per la singola classe Line

Pointcut e Advice

- Un pointcut è definito per catturare il controllo e far eseguire un frammento di codice (advice)

```
// pointcut che cattura il metodo setP1 di Line  
pointcut myChange() : call(void Line.setP1(Point));  
  
// advice che esegue quando il pointcut cattura il flusso  
before() : myChange() {  
    // body dell'advice  
    System.out.println("prima");  
}
```

Pointcut target()

- I pointcut possono fornire valori del contesto
 - Il pointcut `target()` fornisce il riferimento all'oggetto su cui è invocato il metodo catturato dall'altro pointcut specificato
 - Il valore del riferimento servirà agli advice

```

    Oggetto su cui il
    metodo è invocato
    pointcut move(Figure fi) :
        target(fi) &&
        call(void Figure.moveBy(int, int));

    after (Figure f) returning : move(f) {
        <f è di tipo Figure,
        istanza su cui è stato invocato moveBy(>
    }
    
```

Aspetto per display

- Notare il valore che il pointcut fornisce all'advice

```

aspect DisplayUpdating {

    pointcut move(Figure figElt):
        target(figElt) &&
        (call(void Figure.moveBy(int, int)) ||
         call(void Line.setP1(Point)) ||
         call(void Line.setP2(Point)) ||
         call(void Point.setX(int)) ||
         call(void Point.setY(int)));

    after(Figure fe) returning: move(fe) {
        Display.update(fe);
    }
}
    
```

```

class Line {
    private Point p1, p2;
    Point getP1() { return p1; }
    Point getP2() { return p2; }

    void setP1(Point p1) {
        this.p1 = p1;
        Display.update(this);
    }
    void setP2(Point p2) {
        this.p2 = p2;
        Display.update(this);
    }
    void moveBy(int dx, int dy) { ... }
}

class Point {
    private int x = 0, y = 0;
    int getX() { return x; }
    int getY() { return y; }

    void setX(int x) {
        this.x = x;
        Display.update(this);
    }
    void setY(int y) {
        this.y = y;
        Display.update(this);
    }
    void moveBy(int dx, int dy) { ... }
}
    
```

Soluzione OO

- “display updating” non ha un suo modulo
 - Evoluzione difficile
 - Cambiamenti in tutte le classi
 - È necessario tracciare e cambiare tutti i chiamanti
 - Line e Point non dovrebbero saper nulla di Display

```

class Line {
    private Point p1, p2;
    Point getP1() { return p1; }
    Point getP2() { return p2; }

    void setP1(Point p1) {
        this.p1 = p1;
    }
    void setP2(Point p2) {
        this.p2 = p2;
    }
    void moveBy(int dx, int dy) { ... }
}

class Point {
    private int x = 0, y = 0;
    int getX() { return x; }
    int getY() { return y; }

    void setX(int x) {
        this.x = x;
    }
    void setY(int y) {
        this.y = y;
    }
    void moveBy(int dx, int dy) { ... }
}
    
```

Con aspetto

```

aspect DisplayUpdating {

    pointcut move(Figure figElt):
        target(figElt) &&
        (call(void Figure.moveBy(int, int)) ||
         call(void Line.setP1(Point)) ||
         call(void Line.setP2(Point)) ||
         call(void Point.setX(int)) ||
         call(void Point.setY(int)));

    after(Figure fe) returning: move(fe) {
        Display.update(fe);
    }
}
    
```

- Modulo per display updating
 - Il concern è descritto in un singolo modulo
 - Tutti i cambiamenti in un singolo aspetto
 - Evoluzione modulare