
Programação Orientada a Aspectos

Christina von Flach G. Chavez

DCC – UFBA

flach@dcc.ufba.br

Programação Orientada a Aspectos com AspectJ

Parte II

Este curso

↳ Roteiro

- Parte I
Introdução à Orientação a Aspectos

- Parte II
Programação Orientada a Aspectos com AspectJ

- Parte III
Aplicações

- Parte IV
Desenvolvimento de Software Orientado a Aspectos

- Breve Histórico
- AspectJ
 - Sintaxe e Semântica
 - Exemplos

Programação Orientada a Aspectos (POA)

Aspect-Oriented Programming (AOP)

- POA é uma nova metodologia de programação que permite a modularização e composição de interesses transversais.
- **AspectJ**: principal linguagem de programação
 - Realiza os conceitos do modelo de aspectos
 - aspect
 - pointcut
 - advice
 - inter-type declaration
 - join point
 - weaving

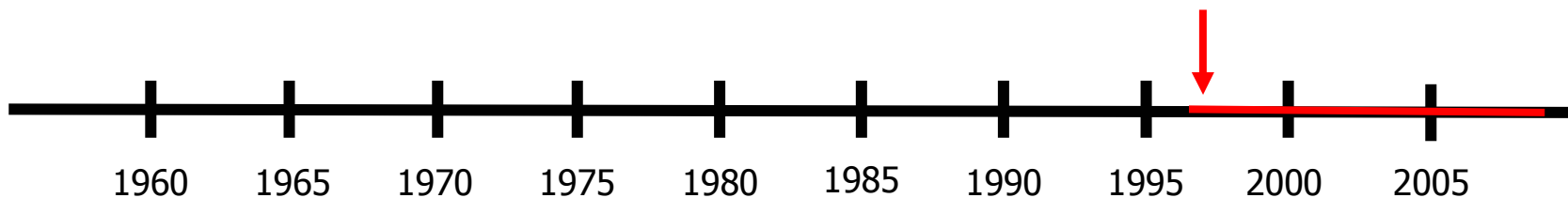
AspectJ™

- AspectJ™ é uma linguagem de programação orientada a aspectos
 - extensão da linguagem Java™
 - de propósito geral

AspectJ

Histórico

- 1996: Aspect-Oriented Programming (AOP)
- 1997: AspectJ, Xerox Parc
- 1998: Primeiros usuários externos
- 2001: **AspectJ 1.0**
- 2002: AspectJ, Eclipse.org
- 2003: AspectJ 1.1.1
- 2004: AspectJ, 1.2
- 2005: AspectJ 5: AspectJ + Aspect Werkz
- 2008: AspectJ 1.6.1



AspectJ

↳ Estágio Atual

- AspectJ 5
 - Nós usaremos a versão 1.2.1 (Novembro, 2004)
 - <http://eclipse.org/aspectj>
- diversos IDEs
 - Eclipse, Emacs, etc.
- *feedback* da constante da comunidade de usuários tem conduzido o projeto da linguagem
 - aspectj-users@eclipse.org

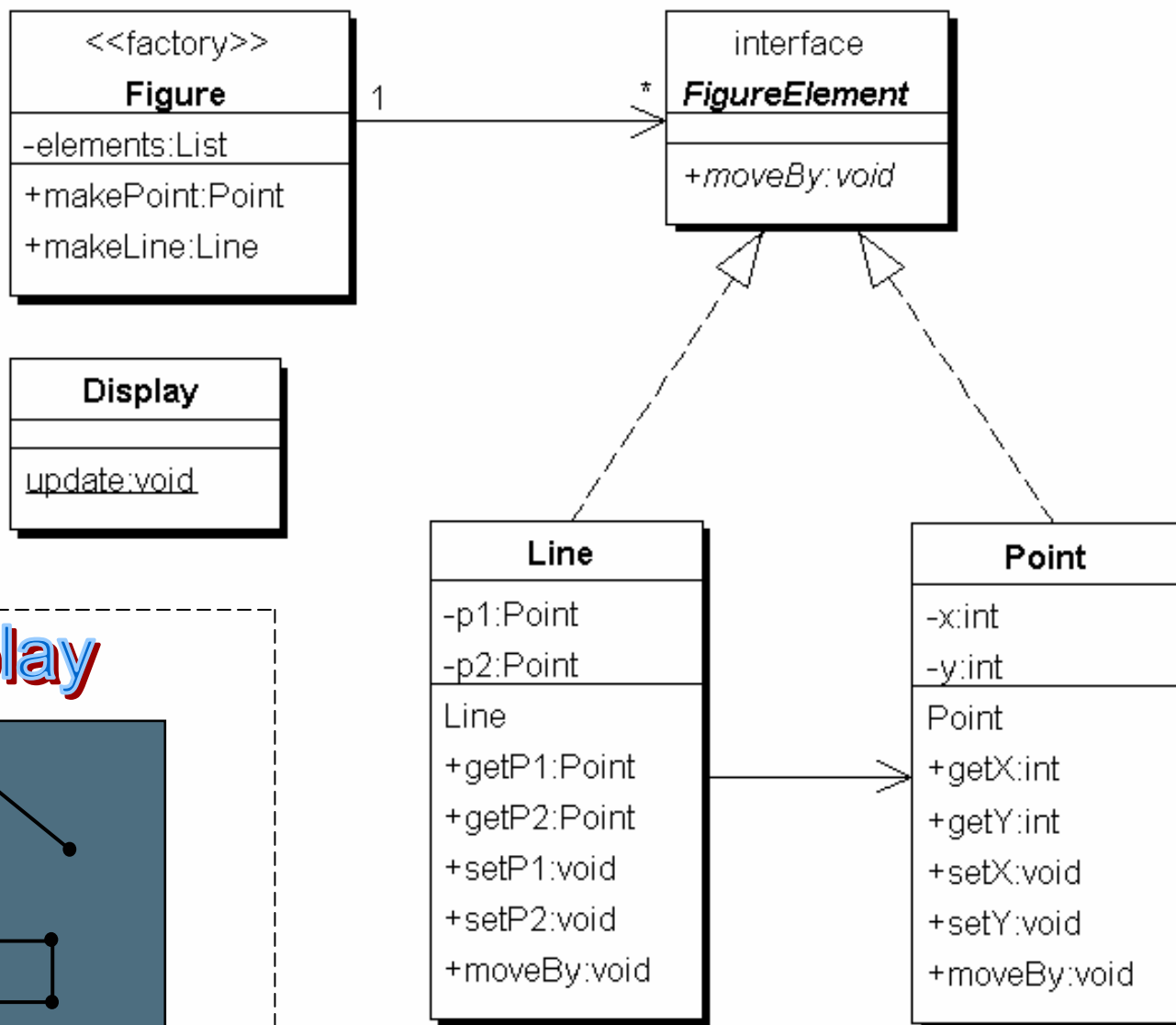
Observação

- Parte do material aqui apresentado baseia-se na documentação disponibilizada livremente pela equipe de AspectJ™:
 - guia de programação
 - palestras e tutoriais diversos
- O material completo pode ser encontrado em:
<http://eclipse.org/aspectj/>

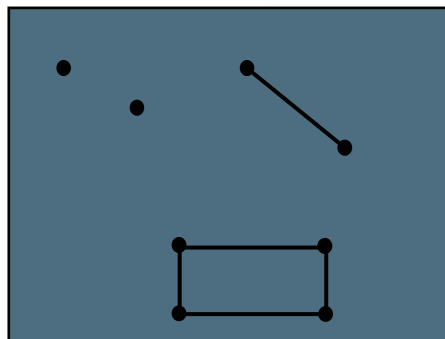
Exemplo

↳ Editor de Figuras

Este exemplo será usado para ilustrar **conceitos básicos** de AspectJ



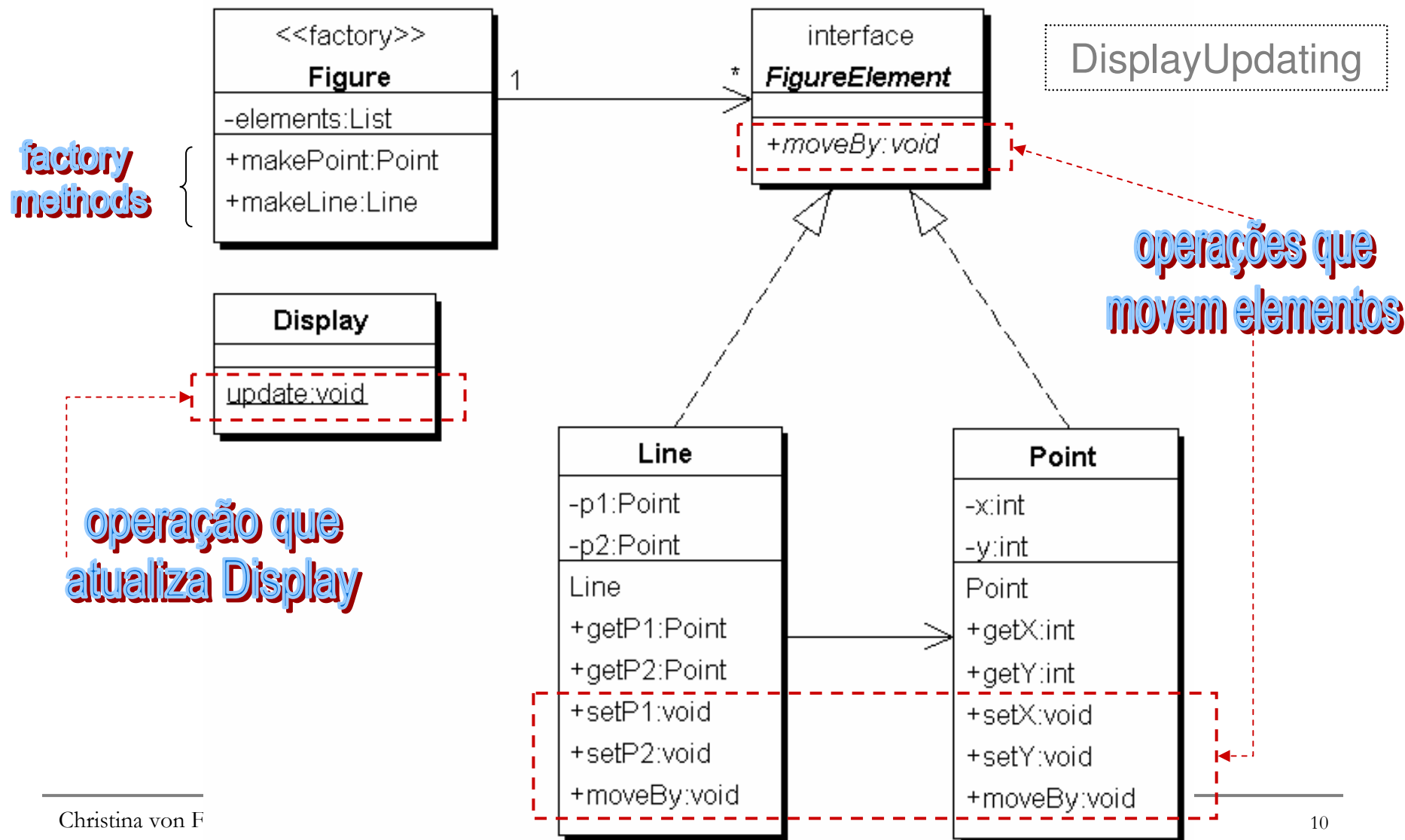
Display



Exemplo

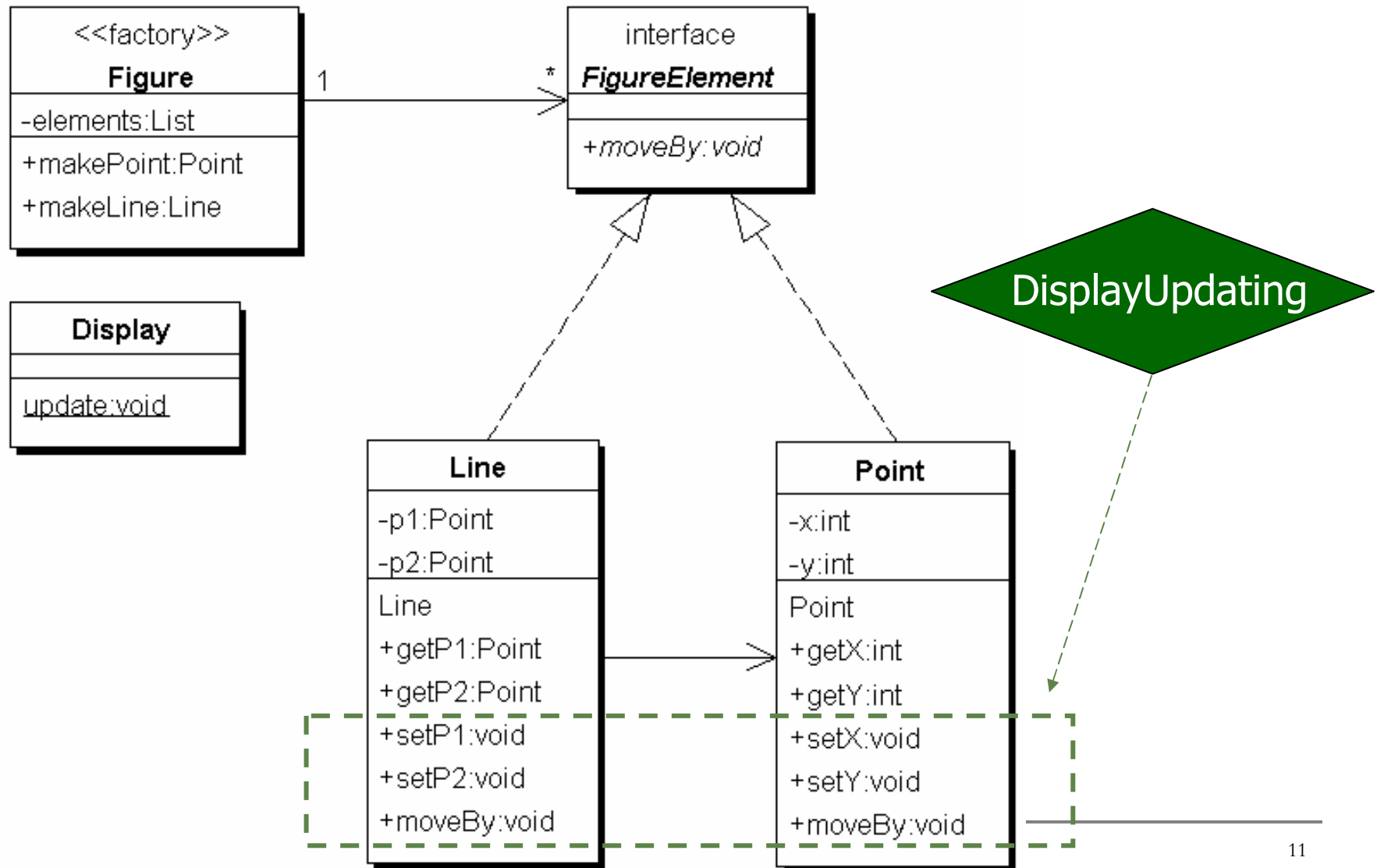
↳ Editor de Figuras

Queremos atualizar Display sempre que elementos forem movimentados



Exemplo

↳ Editor de Figuras



AspectJ

↳ Conceitos Básicos

1. Pontos de junção (*joint points*)
 - pontos na execução de programas Java
2. *Pointcuts*
 - especificação de conjuntos de pontos de junção
3. *Advice*
 - especificação de comportamento que afeta o programa em seus pontos de junção
4. Declarações entre tipos (*inter-type decl., introductions*)
 - declaração de membros e relações entre classes que afetam a estrutura e hierarquia de classes do programa
5. *Aspecto*
 - unidade que modulariza interesses transversais; contém advice, pointcuts, declarações inter-type, atributos, e métodos locais.

Exemplo

```
/**  
 * @author Christina  
 */
```

```
public aspect BoasManeiras {  
    pointcut deliverMessage():  
        call (* MessageCommunicator.deliver(..));  
  
    before(): deliverMessage() {  
        System.out.print("Olá! ");  
    }  
}
```

Exemplo

↳ Definição de DisplayUpdating em AspectJ

- A. identificar os pontos de junção
- B. especificar os pontos de junção

pointcuts

- C. especificar as ações a serem combinadas nos pontos de junção

advice

- D. definir o aspecto (B. and C.)

aspect

Conceitos Básicos

↳ Pontos de junção

- Há diversos pontos identificáveis na **estrutura** de um programa orientado a objetos:
 - classes
 - interfaces
 - relações entre classes
 - atributos
 - métodos
 - comandos
 - ...
- Há diversos pontos identificáveis na **execução** de um programa orientado a objetos:
 - chamada de método
 - execução de método
 - atribuição a variável
 - construção de um objeto
 - execução de comando
 - “return”
 - condicional (if ... then)
 - de repetição (while .. do)
 - levantar uma exceção
 - ...

Conceitos Básicos

↳ Pontos de junção

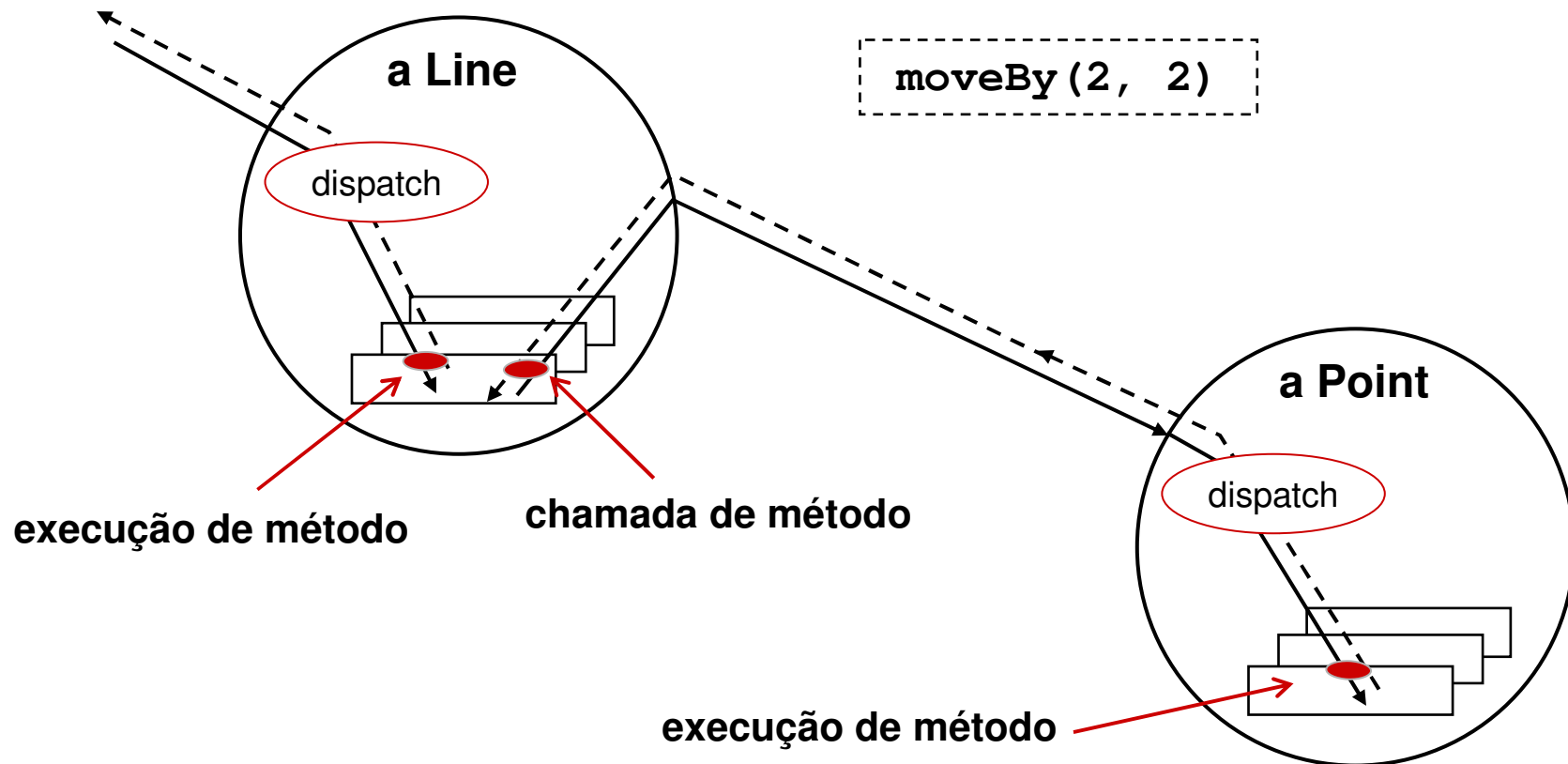
- Em teoria,
 - aspectos podem entrecortar (*crosscut*) **pontos identificáveis** na estrutura ou na execução de programas
- Na prática,
 - AspectJ não permite *crosscutting* arbitrário
 - determina um subconjunto de **pontos de interesse**
- Estes pontos de interesse são chamados de *join points* (pontos de junção)
 - Pontos em que um aspecto AspectJ pode atuar
 - advice

Conceitos Básicos

↳ Pontos de junção

```
public void moveBy(int dx, int dy) {  
    getP1().moveBy(dx, dy);  
    getP2().moveBy(dx, dy);  
} // na classe Line
```

- Em AspectJ, um *ponto de junção* é um ponto bem definido no fluxo de execução de um programa.



Conceitos Básicos

↳ Modelo de pontos de junção

- AspectJ suporta um **modelo dinâmico** de pontos de junção
 - **pontos de junção** são pontos bem-definidos que ocorrem durante a execução de programa Java

- **Alguns pontos de junção**
 - Métodos
 - chamada de métodos
 - execução de métodos
 - Construtores
 - chamada de construtores
 - execução de construtores
 - Execução de tratador de exceções
 - Atributos
 - consulta de valores de atributos
 - modificação de valores de atributos
 - etc.

Conceitos Básicos

↳ Pontos de junção e contexto associado

- Todos os pontos de junção possuem um contexto associado
- Exemplo
 - contexto de uma chamada de método
 - objeto que faz a chamada (*this*)
 - objeto alvo (*target*)
 - argumentos
- Informações de contexto podem ser usadas pelo aspecto

Exemplo

pontos de junção e contexto

```
class Line implements FigureElement {  
    Point p1, p2;  
    ...  
    public void setP1(Point p) {  
        p1 = p;  
    }  
    ...  
}
```

- Execução do método setP1
- Atribuição a p1

Exemplo

↳ Definição de DisplayUpdating

A. Identificar pontos de junção

- ponto de junção dinâmico: execução ou chamada de método

```
class Point implements FigureElement {
    private int x = 0, y = 0;

    Point(int x, int y) {
        super();
        this.x = x;
        this.y = y;
    }

    public int getX() { return x; }
    public int getY() { return y; }

    public void setX(int x) { this.x = x; }
    public void setY(int y) { this.y = y; }

    public void moveBy(int dx, int dy) {
        setX(getX() + dx);
        setY(getY() + dy);
    }
}
```

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

    Line(Point p1, Point p2) {
        super();
        this.p1 = p1;
        this.p2 = p2;
    }

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

    public void setP1(Point p1) { this.p1 = p1; }
    public void setP2(Point p2) { this.p2 = p2; }

    public void moveBy(int dx, int dy) {
        getP1().moveBy(dx, dy);
        getP2().moveBy(dx, dy);
    }
}
```

Pontos de junção relacionados a métodos

- Provavelmente os mais usados
 - chamada de métodos
 - execução de métodos
- Considerar **contexto** a ser exposto e o **escopo** da modificação

Conceitos Básicos

↳ Pointcuts (pontos de corte)

- Um *pointcut* é um tipo de predicado definido sobre pontos de junção que:
 - seleciona pontos de junção (por casamento de padrões), e
 - opcionalmente, expõe valores relativos a esse ponto de junção (informação de contexto)

call (void Point.setX(int))

- *Pointcuts* podem ser:
 - primitivos
 - definidos pelo usuário

casa se o ponto de junção for uma chamada de método (**call**) com essa assinatura

Conceitos Básicos

↳ Pointcuts primitivos

- Pointcuts primitivos **baseados em nomes**
 - especificação baseada nomes de métodos, tipos de parâmetros, etc.)
 - **enumeração explícita de nomes**
- Pointcuts primitivos **baseados em propriedades**
 - especificação baseada em outras propriedades de métodos
 - **uso de *wildcards***

```
call (void Figure.make*(..))
```


Conceitos Básicos

↳ Pointcuts primitivos

- call `call(void Point.setX(int))`
- execution `execution(void Point.setX(int))`
- handler `handler(ArrayOutOfBoundsException)`
- get, set `get(int Point.x)`
- within,
withincode `within(figures.*)`
`withincode(Figure.moveBy(..))`
- cflow, cflowbelow `cflow(call(Figure.moveBy (..))`

- this `this(SomeType)`
- target `target(SomeType)`
- args `args(int,int)`

Conceitos Básicos

↳ Pointcuts primitivos baseados em propriedades

■ Notação

- * denota qualquer tipo e quantidade de caracteres, exceto '.'
- .. denota qualquer quantidade de caracteres incluindo '.'
- + denota qualquer subclasse de um dado tipo

```
call (void Figure+.make*(..))
```

Conceitos Básicos

↳ Pointcuts definidos pelo usuário

nome

parâmetros do pointcut

pointcut **move()**:

call(void FigureElement.moveBy(int, int))

- public, private,...

Conceitos Básicos

↳ Composição de pointcuts

- Pointcuts podem ser compostos usando &&, || e !

```
call(void Line.setP1(Point)) || call(void  
Line.setP2(Point));
```

or

```
pointcut move(Line l):  
target(l) && (call(void Line.setP1(Point)) ||  
call(void Line.setP2(Point)));
```

and

Exemplo

↳ Definição de DisplayUpdating

- B. Especificar pontos de combinação (através de *pointcuts*)

pointcut **move()**:

```
call(void FigureElement.moveBy(int, int)) ||  
  call(void Point.setX(int))                ||  
  call(void Point.setY(int))                ||  
  call(void Line.setP1(Point))              ||  
  call(void Line.setP2(Point))
```

- Este pointcut identifica chamadas de métodos que movem figuras (*figure elements*).

Conceitos Básicos

↳ Advice

- *Advice* define código que será executado nos pontos de junção
 - *before advice*
 - executado quando um ponto de junção é alcançado e **antes** que a computação prossiga
 - *after advice*
 - executado **depois** que a computação realizada no ponto de junção se encerra
 - *around advice*
 - executado quando se chega no ponto de junção; pode possuir (ou não) **controle explícito** sobre a execução da computação originalmente associada ao ponto de junção.
 - *proceed*

Conceitos Básicos

↳ Advice

```
before(): move() {  
    System.out.println("Um elemento vai ser movido.");  
}
```

```
after() returning: move() {  
    System.out.println("Um elemento foi movido. " );  
}
```

```
around() returns void: move(FigureElement) {  
    System.out.println("permiteMovimento == " + enableMove);  
    if (enableMove) { proceed(); }  
    System.out.println("around advice concluído");  
}
```

Exemplo

↳ Definição de `DisplayUpdating`

C. Especificar as ações a serem combinadas

pointcut `move()`:

```
...  
call(void Line.setP1(Point)) ||  
call(void Line.setP2(Point)) ||  
...
```

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

- after returning
- after throwing

Conceitos Básicos

↳ Aspectos

- Um aspecto é a unidade de modularização de interesses transversais de AspectJ
- Um aspecto pode conter ou definir:
 - declarações de *pointcuts*
 - declarações de *advice*
 - declarações intertipos
 - membros (atributos e métodos) locais
 - uma hierarquia de aspectos

Exemplo

DisplayUpdating v1

D

Definição de DisplayUpdating

aspecto

interface

```
aspect DisplayUpdating {
```

```
  pointcut move():
```

```
    call(void FigureElement.moveBy(int, int)) ||  
    call(void Line.setP1(Point))           ||  
    call(void Line.setP2(Point))           ||  
    call(void Point.setX(int))             ||  
    call(void Point.setY(int));
```

pointcut

várias
classes

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

advice

Contexto

DisplayUpdating v2

Conceitos Básicos

↳ Contexto em Pointcuts

- *pointcut* pode expor alguns valores
- *advice* pode usar valor

parâmetro do pointcut

```
pointcut move(FigureElement fe):
```

```
    target(fe) &&
```

```
    (call(void FigureElement.moveBy(int, int)) ||
```

```
    call(void Line.setP1(Point))           ||
```

```
    call(void Line.setP2(Point))           ||
```

```
    call(void Point.setX(int))             ||
```

```
    call(void Point.setY(int)));
```

```
    after(FigureElement fe) returning: move(fe)
```

```
    {    ... fe    ... }
```

parâmetro
do advice

Conceitos Básicos

↳ target

target(<type name>)

- semântica:
 - expõe objeto “alvo”
 - predicado definido sobre ponto de junção
 - qualquer ponto de junção em que o objeto “alvo” é uma instância de <type name>
- Exemplos:
 - target(Point)
 - target(Line)
 - target(FigureElement)
- “qualquer ponto de junção” significa ponto de junção de qualquer tipo (call, execution, etc.)

Conceitos Básicos

↳ Parâmetros

de pointcut

- uma variável é amarrada através de declaração de pointcut definida pelo usuário
 - pointcut fornece valor para a variável
 - valor fica disponível para todos os usuários do pointcut definido pelo usuário

```
pointcut move (Line l) :  
    target (l) &&  
    (call (void Line.setP1 (Point)) ||  
     call (void Line.setP2 (Point)));
```

parâmetro do pointcut

variável

```
after (Line line) : move (line)  
{ ... }
```

Conceitos Básicos

↳ Parâmetros

de advice

- variável é amarrada através de declaração de advice
 - pointcut fornece valor para a variável
 - valor fica disponível no corpo do advice

```
pointcut move(Line l) :  
    target(l) &&  
    (call(void Line.setP1(Point)) ||  
     call(void Line.setP2(Point)));
```


parâmetro do advice


variável declarada


```
after(Line line) : move(line)  
{ ... }
```

Conceitos Básicos

↳ Parâmetros

- O valor se propaga
 - da direita para a esquerda através do ':'
 - lado esquerdo : lado direito 
 - de pointcuts para pointcuts definidos pelo usuário
 - de pointcuts para advice, e depois para corpo do advice

```
pointcut move(Line l):  1  
    target(l) &&  
    (call(void Line.setP1(Point)) ||  
     call(void Line.setP2(Point)));
```

```
after(Line line): move(line) {  2  
    ...  
}
```

Conceitos Básicos

↳ target

target(<type name>)

- semântica:
 - expõe objeto “alvo”
 - predicado definido sobre ponto de junção
 - qualquer ponto de junção em que o objeto “alvo” é uma instância de <type name>
- Exemplos:
 - target(Point)
 - target(Line)
 - target(FigureElement)
- “qualquer ponto de junção” significa ponto de junção de qualquer tipo (call, execution, etc.)

Conceitos Básicos

↳ this

this(<type name>)

- semântica:
 - expõe objeto corrente
 - predicado definido sobre ponto de junção
 - qualquer ponto de junção em que o **objeto corrente** é uma instância de <type name>
- Exemplos:
 - this(Point)
 - this(Line)
 - this(FigureElement)

Conceitos Básicos

↳ args

args(<type name>+)

- semântica:
 - expõe argumentos de chamada ou execução
 - predicado definido sobre ponto de junção:
 - qualquer ponto de junção em que os argumentos passados são instância de
 <type name>, ...
- Exemplos:
 - args(Point)
 - args(Line)
 - args(Line, Point)

Alguns pontos de junção e seus contextos

Ponto de junção	Objeto “corrente” “this”	Objeto “alvo” “target”	Argumentos “args”
Method Call	executing object	target object	method args
Method Execution	executing object	executing object	method args
Constructor Call	executing object	-----	constructor args
Constructor Execution	executing object	executing object	constructor args
Field reference	executing object	target object	-----
Field assignment	executing object	target object	assigned value
Handler execution	executing object	executing object	caught exception

Exemplo

↳ DisplayUpdating

DisplayUpdating v2

```
aspect DisplayUpdating {  
  
    pointcut move(FigureElement fe):  
        target(fe) &&  
        (call(void FigureElement.moveBy(int, int)) ||  
         call(void Line.setP1(Point)) ||  
         call(void Line.setP2(Point)) ||  
         call(void Point.setX(int)) ||  
         call(void Point.setY(int)));  
  
    after( FigureElement fe) returning: move(fe) {  
        Display.update(fe);    // faça algo com fe  
    }  
  
}
```

Relembrando...

↳ sem AspectJ

```
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;
    }
}
```

```
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;
    }
}
```

Exemplo

sem AspectJ

(DisplayUpdating v1)

```
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();
    }
}
```

```
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();
    }
    void setY(int y) {
        this.y = y;
        Display.update();
    }
}
```

Exemplo

↳ sem AspectJ

```
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);
    }
}
```

```
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);
    }
}
```

(DisplayUpdating v2)

- não localiza DisplayUpdating
 - evolução é complicada
 - mudanças em todas as classes
 - tem que identificar e modificar vários pontos

Exemplo

↳ com AspectJ

(DisplayUpdating v1)

```
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;
    }
}
```

```
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;
    }
}
```

```
aspect DisplayUpdating {

    pointcut move():
        call(void FigureElement.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();
    }
}
```

Exemplo

↳ com AspectJ

(DisplayUpdating v2)

```
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;
    }
}

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;
    }
}
```

```
aspect DisplayUpdating {

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

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

- modularização de DisplayUpdating é clara:
 - todas as modificações em um único aspecto
 - evolução modular

Top-level moves

DisplayUpdating v3

Problema

Display.update() chamado mais de uma vez

■ Ex.: linha.moveBy(2,2)

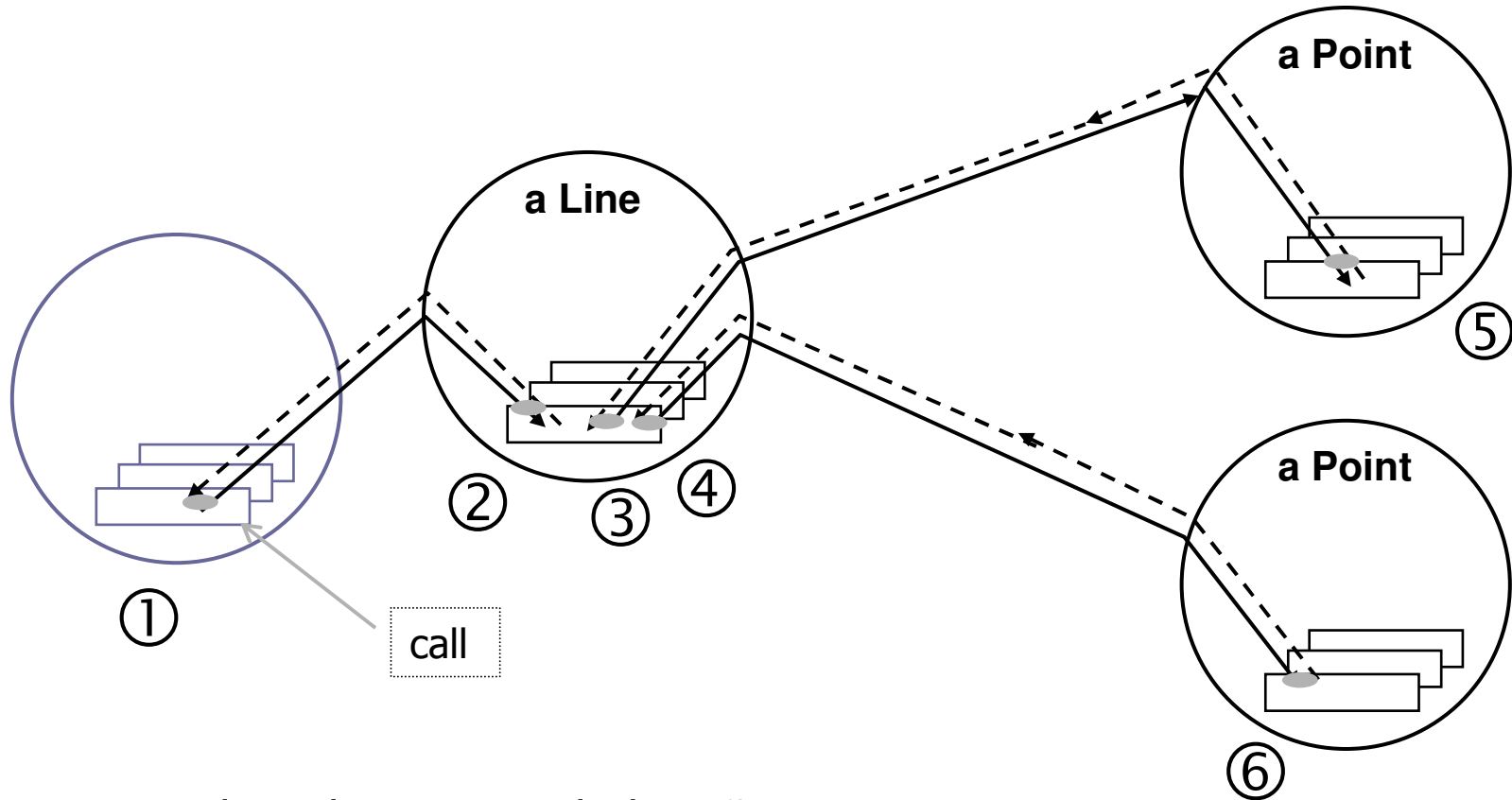
```
public void moveBy(int dx, int dy)
{
    getP1().moveBy(dx, dy);
    getP2().moveBy(dx, dy);
}
```

■ Ex.: ponto.moveBy(2,2)

```
public void moveBy(int dx, int dy)
{
    setX(getX() + dx);
    setY(getY() + dy);
}
```

Conceitos Básicos

↳ Pontos no Fluxo de Controle



os demais pontos de junção aparecem no
fluxo de controle dinâmico de ①

Conceitos Básicos

↳ Pointcuts primitivos

■ cflow

- todos **pontos de junção** no fluxo de controle dinâmico de qualquer ponto de junção descrito pelo pointcut (inclusive)

■ cflowbelow

- todos pontos de junção no fluxo de controle dinâmico abaixo de qualquer ponto de junção descrito pelo pointcut (exceto o próprio)

pointcut topLevelMove (FigureElement fe):

```
move (fe) && !cflowbelow (move (FigureElement));
```

Exemplo

↳ DisplayUpdating

```
aspect DisplayUpdating {  
    pointcut move(FigureElement fe):  
        target(fe) &&  
        (call(void FigureElement.moveBy(int, int)) ||  
         call(void Line.setP1(Point)           ||  
         call(void Line.setP2(Point)           ||  
         call(void Point.setX(int)             ||  
         call(void Point.setY(int)));
```

```
    pointcut topLevelMove (FigureElement fe):  
        move (fe) && !cflowbelow (move (FigureElement) );
```

```
    after(FigureElement fe) returning: topLevelMove(fe) {  
        Display.update(fe);  
    }  
}
```

Um Display por Figura

DisplayUpdating v4

Conceitos Básicos

↳ Declarações *inter-type*

- Aspectos podem alterar a estrutura estática de um programa:
 - adicionando membros (atributos, métodos, etc.) a uma ou mais classes
 - modificando a relação de herança entre classes
 - modificando a relação entre classes e interfaces
- Declarações *intertipos* (ou *inter-type declarations*) implementam *crosscutting estático*
 - a alteração ocorre em tempo de compilação

Um Display por Figura

DisplayUpdating v4

introduzindo atributo
em FigureElement

```
aspect DisplayUpdating {
```

```
    private Display FigureElement.display;
```

```
    static void setDisplay(FigureElement fe, Display d) {  
        fe.display = d;  
    }
```

```
    pointcut move(FigureElement figElt):  
        <como antes>;
```

```
    after(FigureElement fe): move(fe) {  
        fe.display.update(fe);  
    }  
}
```

o atributo **display**

- é atributo de objetos do tipo `FigureElement`
- pertence ao aspecto `DisplayUpdating`
- `DisplayUpdating` define `setDisplay` (setter)

Conceitos Básicos

↳ Extensão e Implementação: declare parents

- Um aspecto pode modificar a hierarquia de herança de um sistema:
 - modificando a superclasse de um tipo
 - declare parents: TypePattern extends TypeList;
 - adicionando uma interface a um tipo
 - declare parents: TypePattern implements TypeList;

```
aspect A {  
    declare parents: SomeClass implements Runnable;  
    public void SomeClass.run() { ... }  
}
```

↑

Aspectos abstratos

DisplayUpdating v5

Conceitos Básicos

↳ Herança e especialização

- pointcuts podem ter advice adicional
 - aspecto com
 - pointcut concreto
 - sem advice definido para o pointcut
 - em Editor de Figuras
 - move() pode ter advice de vários aspectos
 - módulo pode expor pointcuts bem-definidos
- pointcuts abstratos podem ser especializados
 - aspecto com
 - pointcut abstrato
 - advice concreto definido sobre pointcut abstrato

Exemplo

↳ Aspecto e Pointcut abstratos

```
abstract aspect Observing {  
  
    protected interface Subject { }  
    protected interface Observer { }  
  
    public void addObserver(Subject s, Observer o) { ... }  
    public void removeObserver(Subject s, Observer o) { ... }  
  
    abstract pointcut changes(Subject s);  
  
    after(Subject s): changes(s) {  
        Iterator iter = getObservers(s).iterator();  
        while ( iter.hasNext() ) {  
            notifyObserver(s, ((Observer)iter.next()));  
        }  
    }  
    abstract void notifyObserver(Subject s, Observer o);  
}
```

Exemplo

DisplayUpdating v5

↳ Aspecto e Pointcut concretos

```
aspect DisplayUpdating extends Observing {  
  
    declare parents: FigureElement implements Subject;  
    declare parents: Display           implements Observer;  
  
    pointcut changes(Subject s):  
        call(void FigureElement.moveBy(int, int)) ||  
        call(void Line.setP1(Point))           ||  
        call(void Line.setP2(Point))           ||  
        call(void Point.setX(int))             ||  
        call(void Point.setY(int));  
  
    void notifyObserver(Subject s, Observer o) {  
        ((Display)o).update();  
    }  
}
```

Declarações

Conceitos Básicos

↳ Precedência entre aspectos

- Um aspecto pode declarar explicitamente uma relação de precedência entre aspectos concretos:

- `declare precedence : TypePatternList`

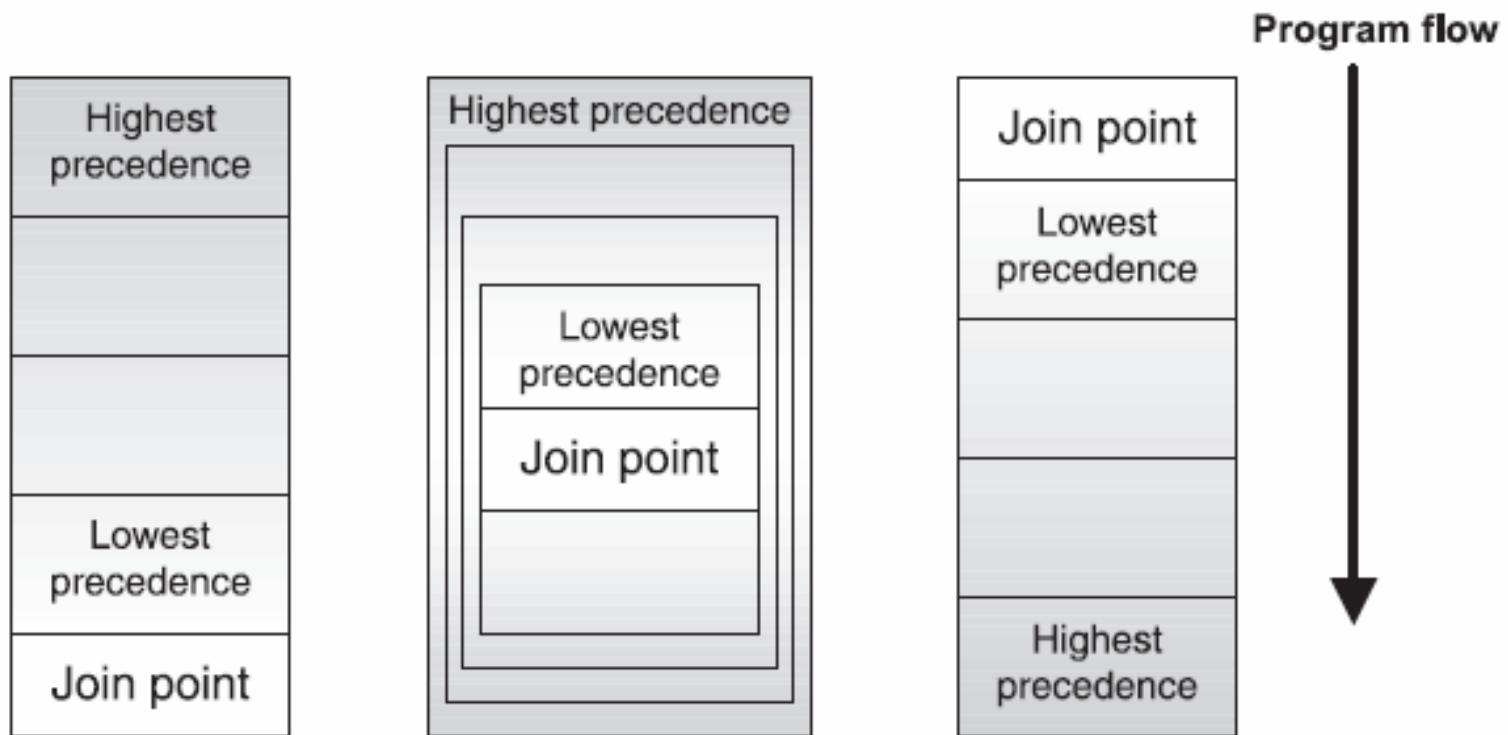
- Exemplo

- `declare precedence: Rastreamento, DisplayUpdate;`

- Rastreamento tem a maior precedência

Conceitos Básicos

↳ Precedência entre aspectos



Conceitos Básicos

↳ Erros e warnings em tempo de compilação

- Um aspecto pode declarar erros e *warnings* a serem verificados em tempo de compilação associados a pontos de junção:
 - declare error
 - declare warning

declare warning :

call(void Persistence.save(Object)) :

"Consider using Persistence.saveOptimized()";

Acesso Reflexivo a Join Points

PointCoordinateTracing

Conceitos Básicos

↳ Acesso reflexivo a informações em ponto de junção

- AspectJ provê elementos de linguagem para dar acesso a informação estática e dinâmica associada a pontos de junção
 - Uso de reflexão
- Exemplo
 - Acesso a nomes de argumentos e nome de métodos afetados por advice
- O contexto dinâmico que poderia ser capturado é similar àquele capturado por **this()**, **target()** e **args()**

Conceitos Básicos

↳ Acesso reflexivo a informações em ponto de junção

- AspectJ oferece acesso reflexivo através de 3 objetos especiais
 - `thisJoinPoint`
 - `thisJoinPointStaticPart`
 - `thisEnclosingJoinPointStaticPart`
- Estes objetos podem ser usados no corpo de qualquer advice

Conceitos Básicos

↳ Acesso reflexivo a informações em ponto de junção

- Informação dinâmica
 - Aquela que deve mudar a cada invocação distinta de um mesmo ponto de junção
 - `Point.setX(int)`
- Informação estática
 - Aquela que não se modifica em tempo de execução
 - nome e o local onde um método está definido

Conceitos Básicos

↳ Acesso reflexivo a informações em ponto de junção

- Cada ponto de junção provê
 - um objeto que contém informação dinâmica
 - **thisJoinPoint**
 - dois objetos que contêm informação estática sobre (a) o ponto de junção e (b) o contexto que envolve o ponto de junção ("enclosing context")
 - **thisJoinPointStaticPart**
 - **thisEnclosingJoinPointStaticPart**

Conceitos Básicos

↳ thisJoinPoint

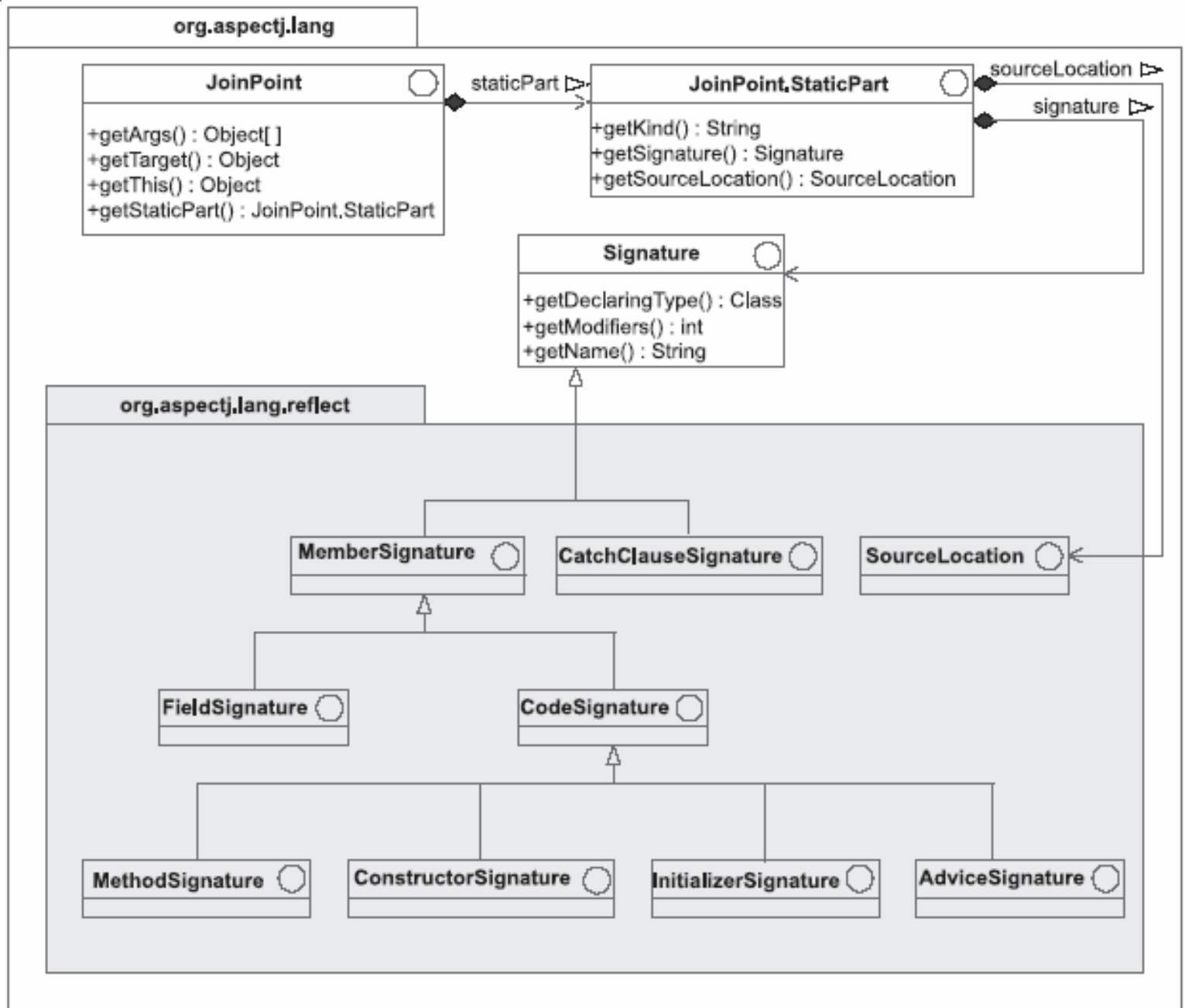
- thisJoinPoint
- acesso reflexivo a informação dinâmica relativa ao ponto de junção:
 - this, target, args
- acesso reflexivo a informação estática relativa ao ponto de junção
 - Uso do método `getStaticPart()`
- Exemplo
 - Guardar valores de objeto corrente e argumentos através de aspecto de Rastreamento

Conceitos Básicos

↪ `thisJoinPointStaticPart` e `thisEnclosingJoinPointStaticPart`

- Acesso a **informação estática** sobre o ponto de junção afetado pelo advice
- `thisJoinPointStaticPart`
 - Acesso ao tipo de ponto de junção (method-call, method-execution, field-set, field-get, ...), à assinatura do método
 - Exemplo
 - Guardar nome dos métodos executados através de aspecto de Rastreamento
- `thisEnclosingJoinPointStaticPart`
 - Acesso ao contexto que envolve o ponto de junção
 - Depende do tipo de ponto de junção
 - Exemplo
 - Guardar informação do contexto, do método que faz a chamada relativa a um **call**, ou do método que contém um bloco **catch**

API



Exemplo

↳ PointCoordinateTracing

```
aspect PointCoordinateTracing {  
  
    before(Point p, int newVal):  
        set(int Point.*) && target(p) && args(newVal) {  
  
        System.out.println("At " +  
            thisJoinPointStaticPart.getSignature() +  
            " field is set to " + newVal + ".");  
  
        }  
    }  
}
```

- set(int Point.*)
 - Qualquer coordenada (x ou y)

Pontos de junção para acesso

- Os pontos de junção para acesso a atributos
 - acesso para leitura (**get**)
 - acesso para escrita (**set**)
- Os acessos considerados são para instâncias ou membros de uma classe
 - Não estão definidos para variáveis locais a métodos

**Associação de Aspectos
Instâncias**

PublicErrorLogging

Associação de aspectos

- Por default, apenas uma instância de aspecto existe para cada JVM
 - como um singleton
- Pode-se desejar uma instância de aspecto diferente associada a cada objeto, fluxo de controle, etc.
- Categorias de associação
 - Por VM (default)
 - Por objeto
 - Por fluxo de controle

Conceitos Básicos

↳ Instâncias de aspectos

- `pertarget(<pointcut>)`
`perthis(<pointcut>)`
 - uma instância de aspecto associada a cada objeto que seja representado por “target” ou “this” nos pontos de junção descritos por `<pointcut>`
- `percflow(<pointcut>)`
`percflowbelow(<pointcut>)`
 - uma instância de aspecto associada a cada ponto de junção em `<pointcut>`, está disponível para todos os pontos de junção de `cflow` ou `cflowbelow`

Exemplo

↳ Instâncias de aspectos

```
aspect PublicErrorLogging
  pertarget(PublicErrorLogging.publicInterface()) {

  Log log = new Log();

  pointcut publicInterface():
    call(public * com.xerox..*.*(..));

  after() throwing (Error e): publicInterface() {
    log.write(e);
  }
}
```

uma instância de aspecto para cada objeto que seja “target” nestes pontos

Conceitos Básicos

↳ Acesso a Instâncias de aspectos

```
...  
static Log getLog(Object obj) {  
    return (PublicErrorLogging.aspectOf(obj)).log;  
}
```



- aspectOf
 - método estático de aspectos
 - devolve instância de aspecto

Pointcuts baseados na estrutura léxica

- within
- withincode

Exemplo

```
aspect FactoryEnforcement {  
  
    pointcut illegalNewFigElt():  
        call(FigureElement+.new(..)) &&  
        !withincode(FigureElement+ Figure.make*(..));  
  
    declare error: illegalNewFigElt():  
        "Illegal figure element constructor call.";  
  
}
```

Resumo

join points

method & constructor
call
execution
field
get
set
exception handler
execution
initialization

aspects

crosscutting type
pertarget
perthis
percflow
percflowbelow

pointcuts

-primitive-

call
execution
handler
get set
initialization
this target
within withincode
cflow cflowbelow

-user-defined-

pointcut declaration
abstract
overriding
static

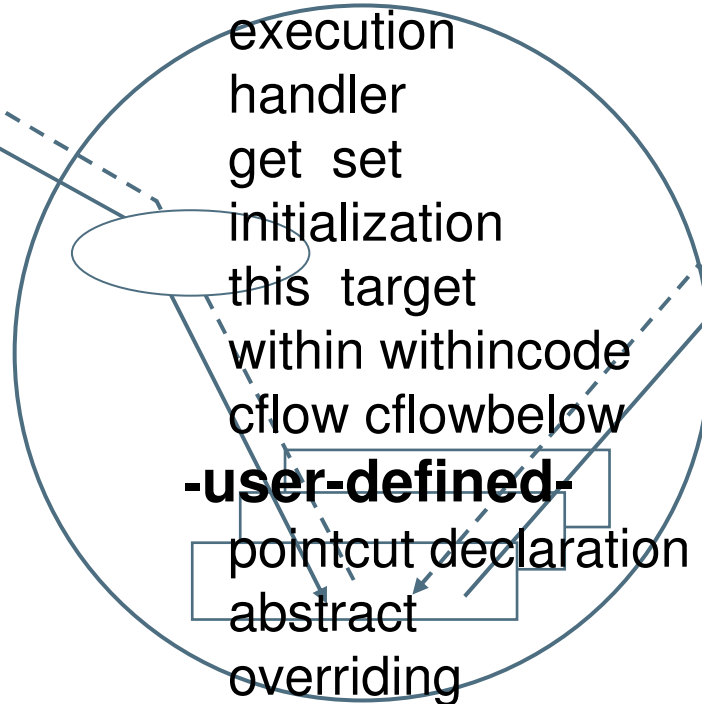
advice

before
after
around

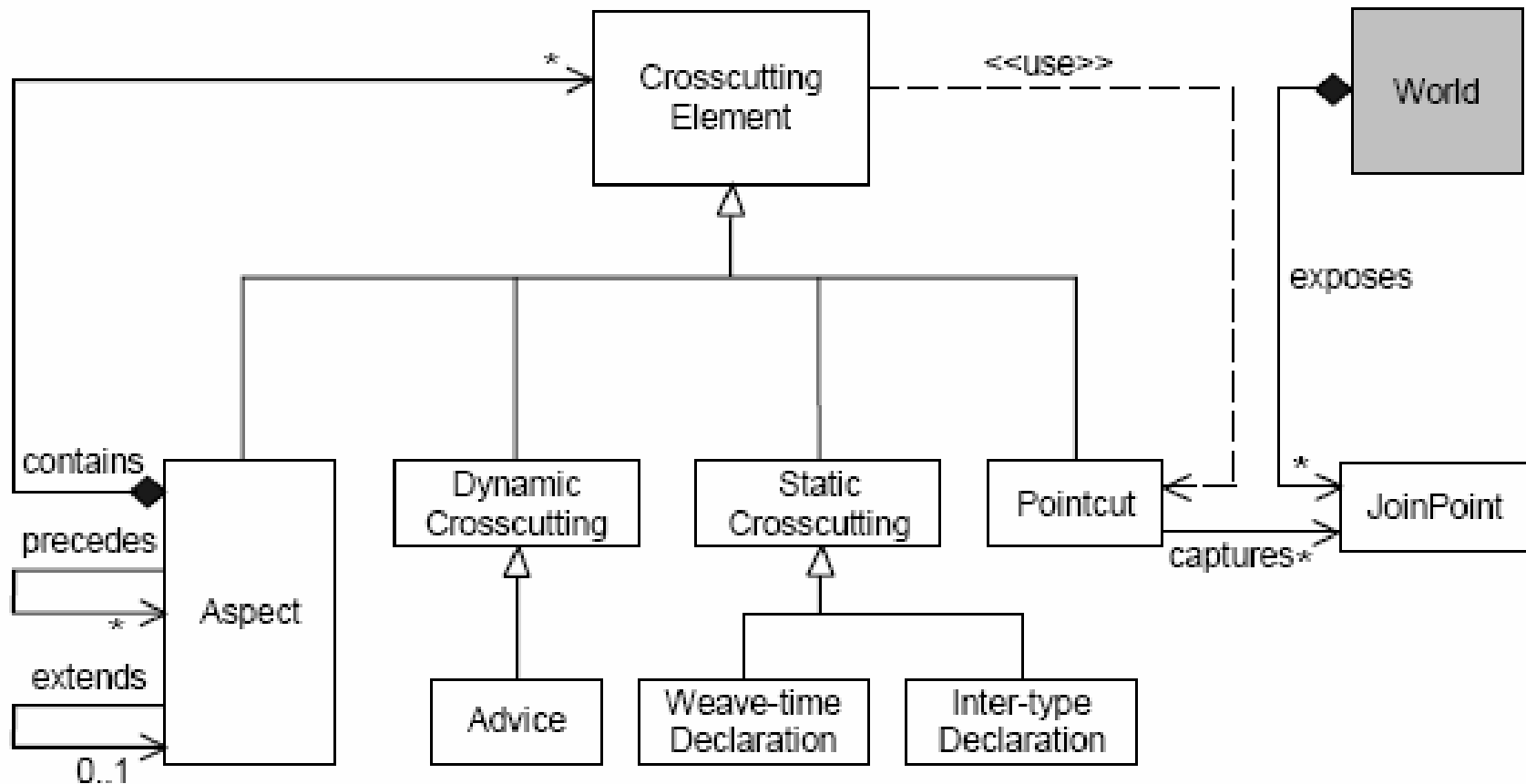
inter-type decls

declare

error
parents
precedence



Modelo de aspectos



Demonstração Eclipse

Editor de Figuras
Exemplos

Final da Parte II
