# Programação Orientada a Aspectos

**Christina von Flach G. Chavez**
**DCC – UFBA**
**flach@dcc.ufba.br**

# Este curso

- Programação orientada a aspectos?

# Programação Orientada a Aspectos (POA)

- Orientação a aspectos
  - qual a motivação?
  - quais os principais conceitos?
    - o que é um aspecto?
  - quais os benefícios?
  - é um novo paradigma?
    - aspectos substituem objetos?

- Programação orientada a aspectos
  - quais as linguagens, ferramentas?
  - aplicações?
    - posso usar aspectos no código de um sistema existente?

- Desenvolvimento de software
  - Como aspectos são tratados em outras atividades do processo de software?

# 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

- Princípios de Engenharia de Software
- Orientação a Objetos
  - Modelo de Objetos
  - Benefícios e limitações
- Orientação a Aspectos
  - Modelo de Aspectos
  - Benefícios e desafios

# 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
- Ambiente de programação

# 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

- Aplicações
  - aspectos de desenvolvimento
  - aspectos de produção
  - aspectos de reutilização
- Exercícios

# 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

- Histórico e Antecedentes
- Programação
- Projeto e Modelagem
- Requisitos
- Métricas
- Tendências

# Este curso

## ↳ Perfil dos Participantes

- Experiência com o tópico curso
  - Programação orientada a objetos
    - Java
  - Reflexão computacional?
  - Programação orientada a aspectos?
    - AspectJ?
    - Outra?

# Introdução à Orientação a Aspectos

## Parte I

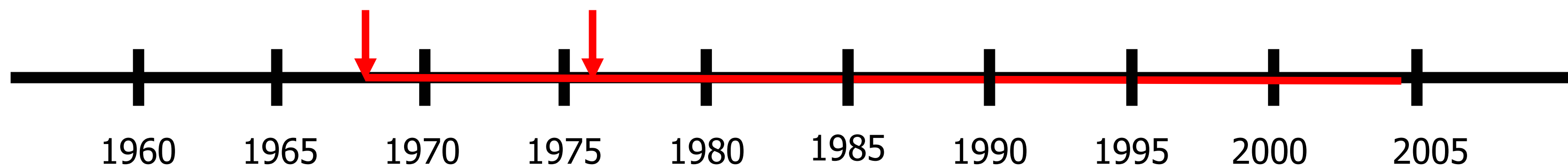
# Engenharia de Software

- A complexidade crescente dos sistemas de software gera novos desafios para as metodologias da Engenharia de Software

- Queremos desenvolver software que seja fácil de:
  - compreender
  - modificar
  - evoluir
  - reutilizar

# Engenharia de Software
## Princípios

- Separação de Interesses (*separation of concerns*)
  - *para contornar a complexidade de um problema, deve-se resolver uma questão importante ou interesse (concern) por vez [Dijkstra 76].*



# Interesses típicos em sistemas de software

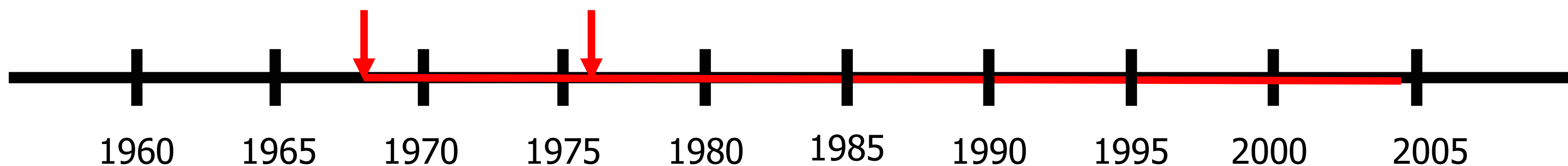
- Funcionalidade básica do sistema
  - "lógica do négocio"
- Propriedades sistêmicas
  - Desempenho
  - Persistência e integridade dos dados
  - Distribuição
  - Segurança
  - Autenticação
  - Auditoria
  - Tratamentos de erros
  - Usabilidade
  - ...

## Decomposição e Modularidade

- *sistemas de software complexos devem ser dividos e organizados em unidades modulares e claramente separadas, com interfaces bem definidas, cada uma lidando com um único interesse.*

## Critério de decomposição

- a eficácia da "modularização" também depende dos critérios usados para a decomposição do sistema em um conjunto de módulos [Parnas 72].



# Decomposição e Modularidade

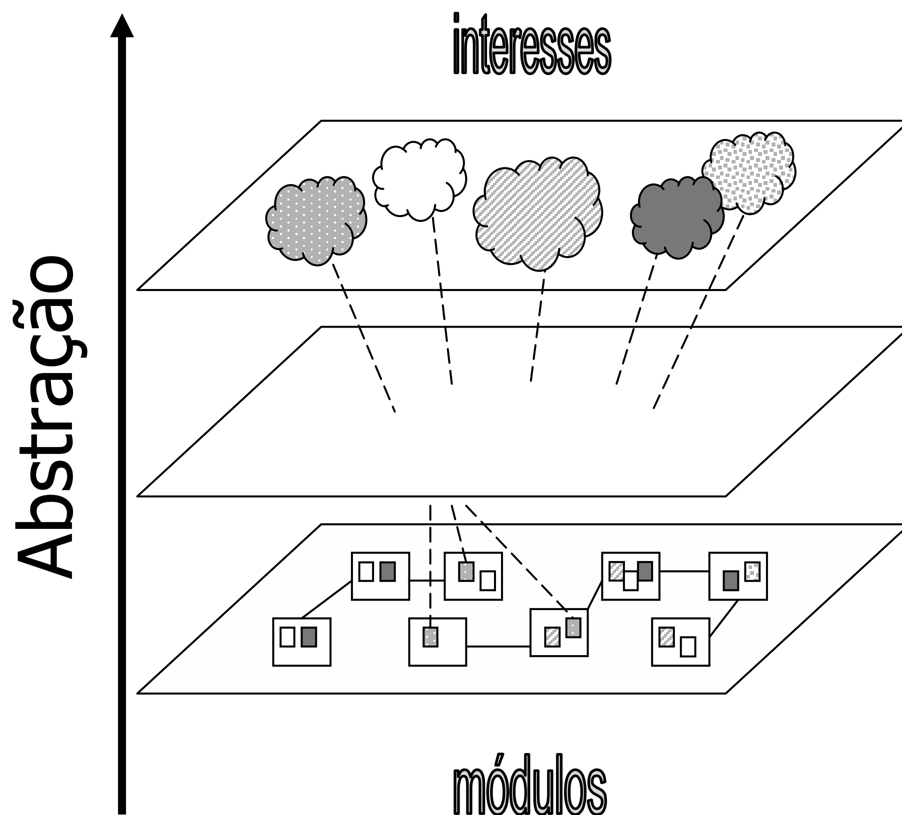
- "Unidades modulares" típicas
  - módulos ou pacotes
  - funções
  - procedimentos
  - tarefas
  - classes
  - agentes
  - ...

- Critérios de decomposição típicos
  - algorítmico
  - orientado a objetos
  - orientado agentes
  - ...

tipo de decomposição predominante

# Desenvolvimento de Software

## ↳Separação de Interesses e Modularidade

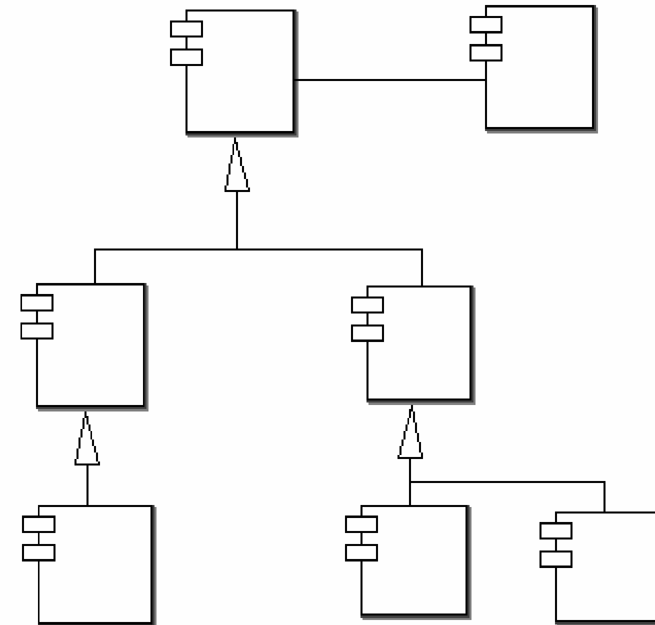



- Identificação de vários interesses
- Associação de interesses a "módulos"
  - acoplamento (-)
  - coesão (+)
- em vários níveis

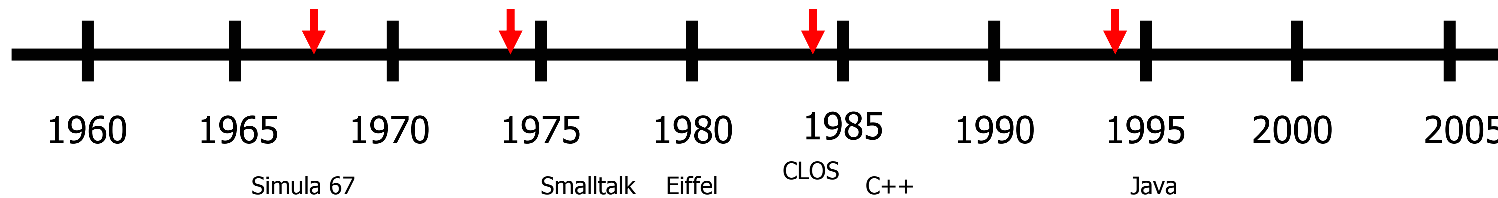
  (requisitos, análise, projeto e implementação)
  - rastreabilidade

# Paradigma de Orientação a Objetos (OO)

- decomposição orientada a objetos
- modularização
  - objetos
  - classes
  - dados e procedimentos encapsulados



problema: ?



# Orientação a Objetos
## ↳O Modelo de Objetos

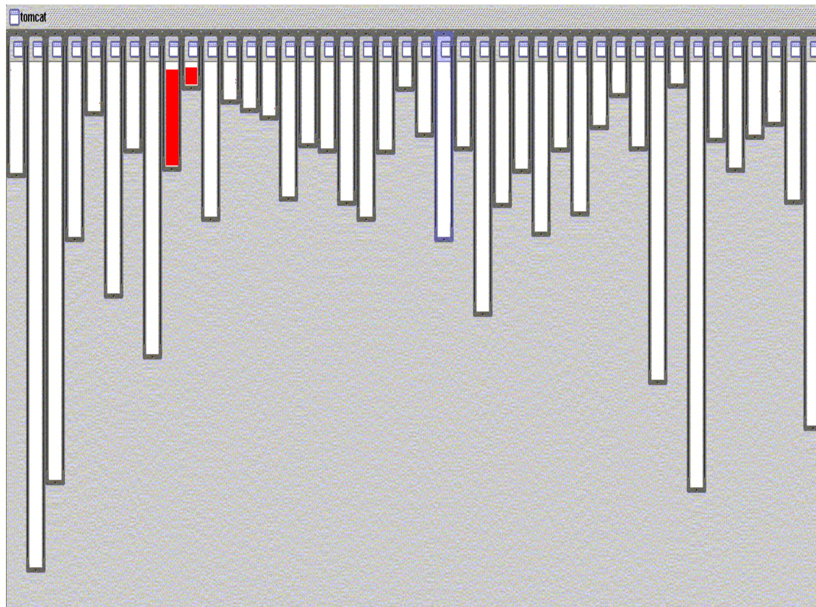
- Arcabouço conceitual
  - classes, objetos
  - herança, agregação
  - troca de mensagens
  - polimorfismo...
- Princípios [Booch]
  - Abstração
  - Encapsulamento
  - Hierarquia
  - Modularidade
  - OCP, ISP, ...
- Requisitos básicos [Cardelli & Wegner]
  - objetos são abstrações de dados
  - objetos possuem uma classe associada
  - classes podem herdar propriedades de superclasses

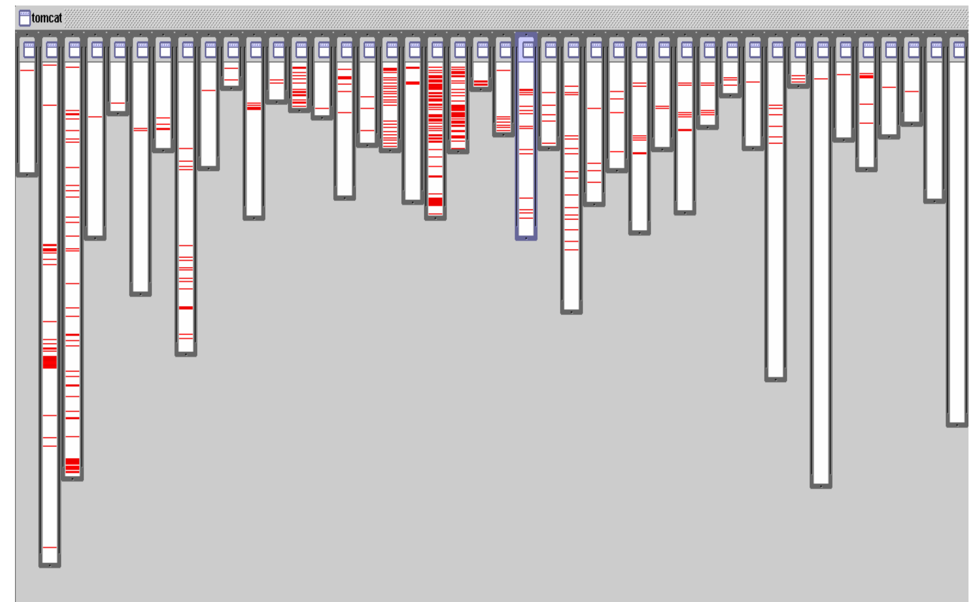
# Orientação a Objetos

- Benefícios
  - Expressividade +
  - Legibilidade +
  - Suporte à evolução +
  - Potencial de reutilização +
  - …
- Limitações
  - *Evolution problems in object-oriented modeling*, [TRESE Group]
  - *Objects have failed* [Dick Gabriel]
  - *OOP Criticism* [B. Jacobs]
  - …
  - dificuldade em modularizar alguns tipos de interesses

# Limitações de OO

## ↳Dificuldade em modularizar alguns tipos de interesses



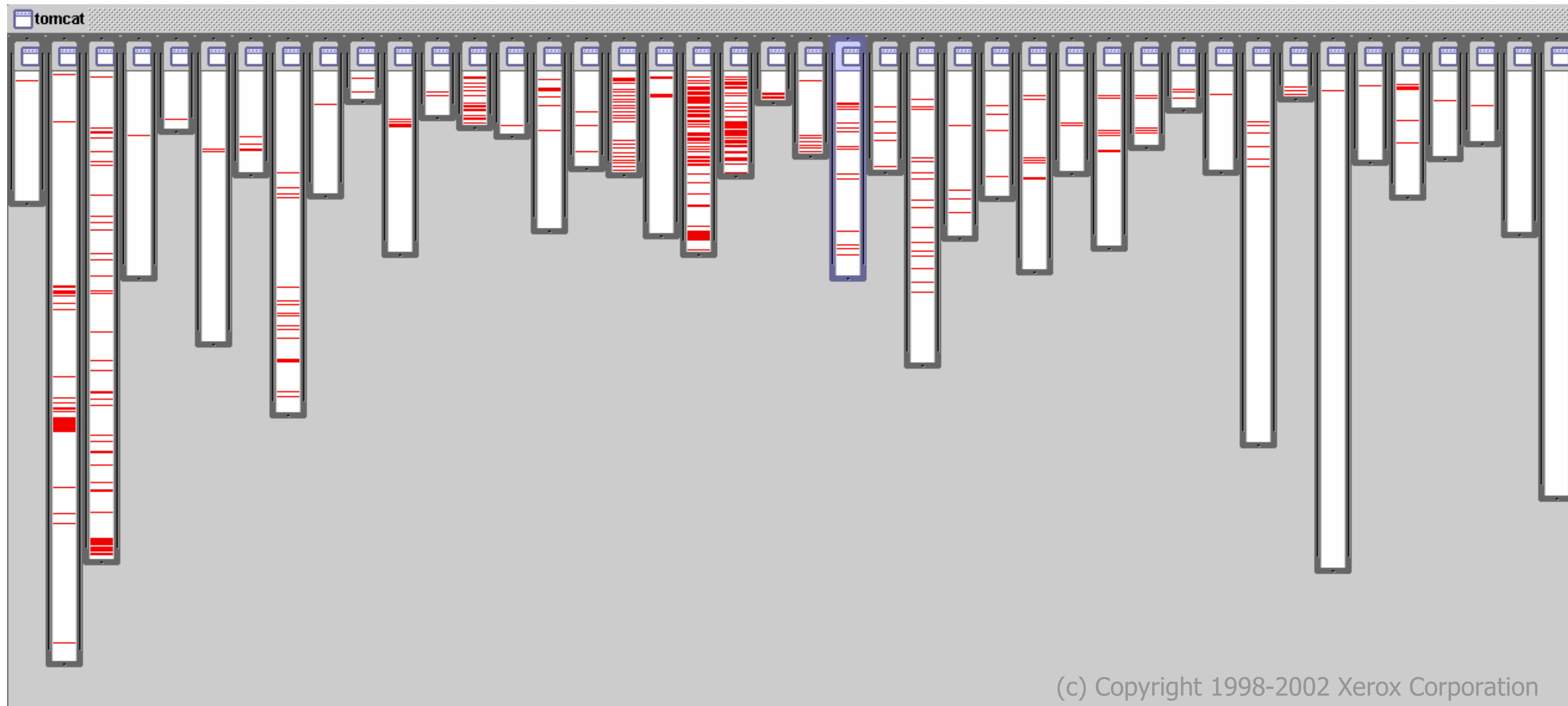




(c) Copyright 1998-2002 Xerox Corporation

- linhas de código relevantes para a implementação da funcionalidade ***P***
- linhas de código relevantes para a implementação da funcionalidade ***R***

# Limitações de OO

## Dificuldade em modularizar alguns tipos de interesses

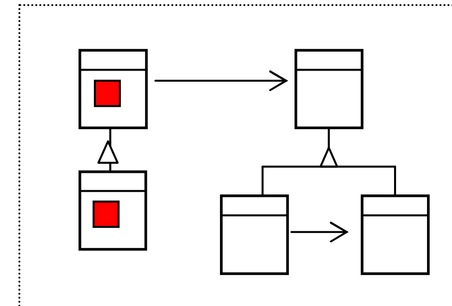



- linhas de código relevantes para a implementação da funcionalidade ***R***
  - aparecem em vários locais
  - aparecem misturadas a linhas de código de outras funcionalidades

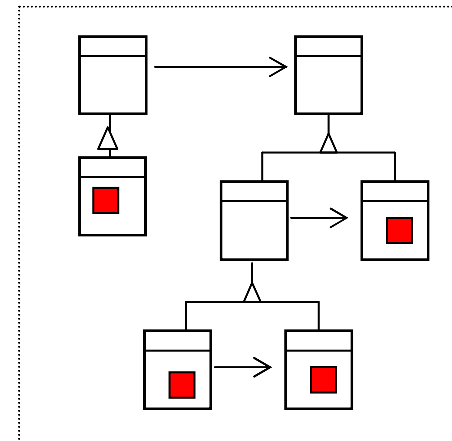
# Limitações de OO

## Dificuldade em modularizar alguns tipos de interesses

- A decomposição em classes modulariza bem a funcionalidade P
  - P aparece em duas classes



- A decomposição em classes não modulariza bem a funcionalidade R
  - R fica espalhada pela hierarquia de classes
  - R fica misturada a outras funcionalidades



# Exemplo
# Sistema Disque Saúde

- Requisitos funcionais
  - Registra e encaminha queixas para o sistema de saúde
  - Exibe informações sobre unidades de saúde e suas especialidades
- Requisitos não-funcionais
  - Distribuído
  - Acesso concorrente
  - Extensível
    - Armazenamento de dados
    - Tecnologia de distribuição

S.Soares, E. Laureano, P. Borba. "**Implementing Distribution and Persistence Aspects with AspectJ**". Proceedings of OOPSLA'02, November 2002.

# Exemplo: Disque Saúde
# Distribuição com RMI

```
public class Complaint implements java.io.Serializable {
    private String description;
    private Person complainer; ...
    public Complaint(String description, Person complainer,
...) {
        ...
    }
    public String getDescription() {
        return this.description;
    }
    public Person getComplainer() {
        return this.complainer;
    }
    public void setDescription(String desc) {
        this.description = desc;
    }
    public void setComplainer(Person complainer) {
        this.complainer = complainer;
    } ...
}
```

```
public class HealthWatcherFacade implements IFacade {
    public void update(Complaint complaint)
            throws TransactionException, RepositoryException,
            ObjectNotFoundException, ObjectNotValidException {
        ...
    }
    public static void main(String[] args) {
        try {
            HealthWatcherFacade facade = HealthWatcherFacade.getInstance();
            System.out.println("Creating RMI server...");
            UnicastRemoteObject.exportObject(facade);
            java.rmi.Naming.rebind("/HealthWatcher");
            System.out.println("Server created and ready.");
        }
        catch (RemoteException rmiEx) {... }
        catch (MalformedURLException rmiEx) { ...}
        catch(Exception ex) {... }
    }
}
```

```
public class ServletUpdateComplaintData extends HttpServlet {
    private IFacade facade;
    public void init(ServletConfig config) throws ServletException {
        try {
            facade = (IFacade) java.rmi.Naming.lookup("//HealthWatcher");
        }
        catch (java.rmi.RemoteException rmiEx) {...}
        catch (java.rmi.NotBoundException rmiEx) {...}
        catch (java.net.MalformedURLException rmiEx) {...}
    }

    public void doPost(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        ...
        facade.update(complaint);
        ...
    } ...
}
```

```
public class Person implements java.io.Serializable {
    private String nome; ...
    public Person(String nome, …) {
        this.nome = nome; …
    }
    public String getNome() {
        return nome;
    } …
}
```

```
public interface IFacade extends java.rmi.Remote {

    public void updateComplaint complaint)
            throws TransactionException, RepositoryException,
            ObjectNotFoundException, ObjectNotValidException,
            RemoteException;
  . . .
}
```

Código RMI

S.Soares, E. Laureano, P. Borba. "**Implementing Distribution and Persistence Aspects with AspectJ**". Proceedings of OOPSLA'02, November 2002.

# Exemplo

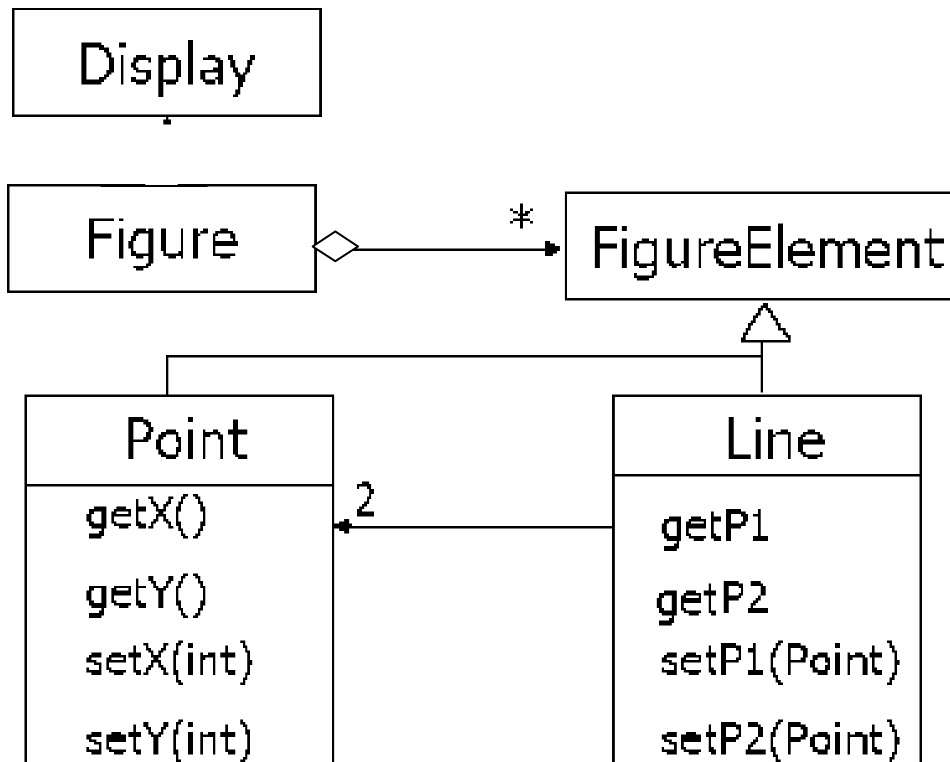
## Editor de Figuras

- Uma figura (Figure) é composta de vários elementos (FigureElement). Um elemento pode ser um Ponto (Point) ou uma Linha (Line). Um ponto é definido através de suas coordenadas x e y. Uma linha é definida a partir de dois pontos p1 e p2.
- Figuras são exibidas em uma tela (Display)

- Projeto OO usando UML
  - Observe os princípios de separação de interesses, decomposição, modularidade, encapsulamento
  - Assegure-se que os módulos são coesos e fracamente acoplados.

# Exemplo

## Editor de Figuras




- módulos = classes
  - coesos
  - fracamente acoplados
  - interface bem definidas

# Exemplo

## Editor de Figuras

```
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;

  Point getX() { return x; }
  Point getY() { return y; }

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

# Exemplo

## ↳Editor de Figuras

- Problema I: DisplayUpdating
  - atualizar a informação na tela (Display) sempre que um elemento (Figure, Line ou Point) for modificado.
- Identificação
  - métodos que modificam elementos

    void Point.setX(int), void Point.setY(int)

    void Line.setP1(Point), void Line.setP2(Point)
  - método que atualiza informação na tela

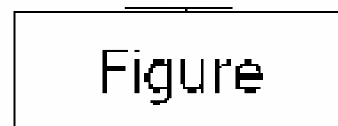
    Display.update()
- Solução

  **?**

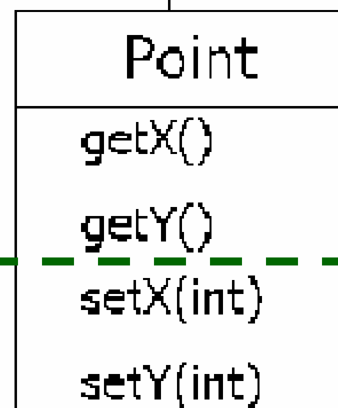
# Exemplo

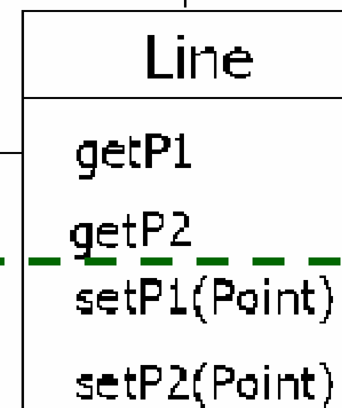
## ↳Editor de Figuras

operação que
atualiza a tela

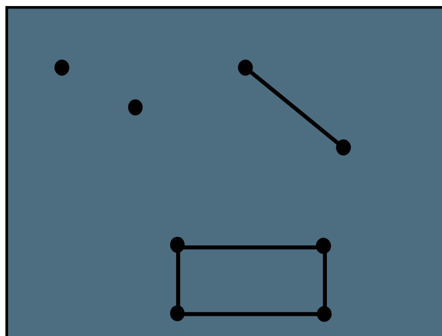


operações que
modificam elementos




Display



# Exemplo (DisplayUpdating)

## Editor de Figuras

```
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();
  }
}
```

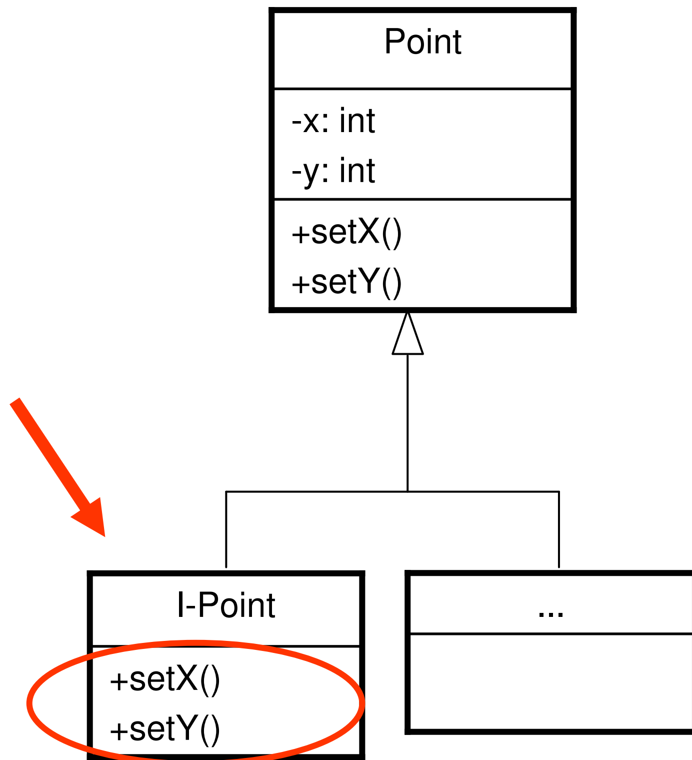
```
class Point {
  private int x=0, y=0;

  Point getX() { return x; }
  Point getY() { return y; }

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

- Modificação invasiva
  - chamadas ao método **update()** estão espalhadas e misturadas no código

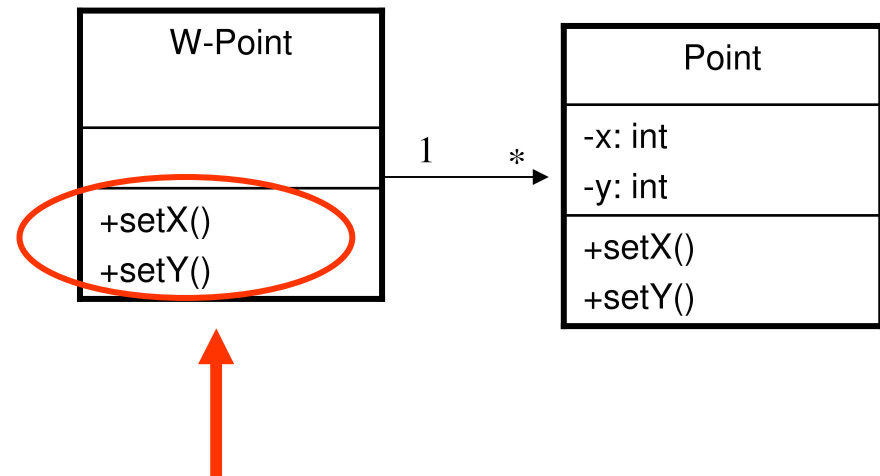
# Exemplo

## Outras opções

- Extensão através de Herança

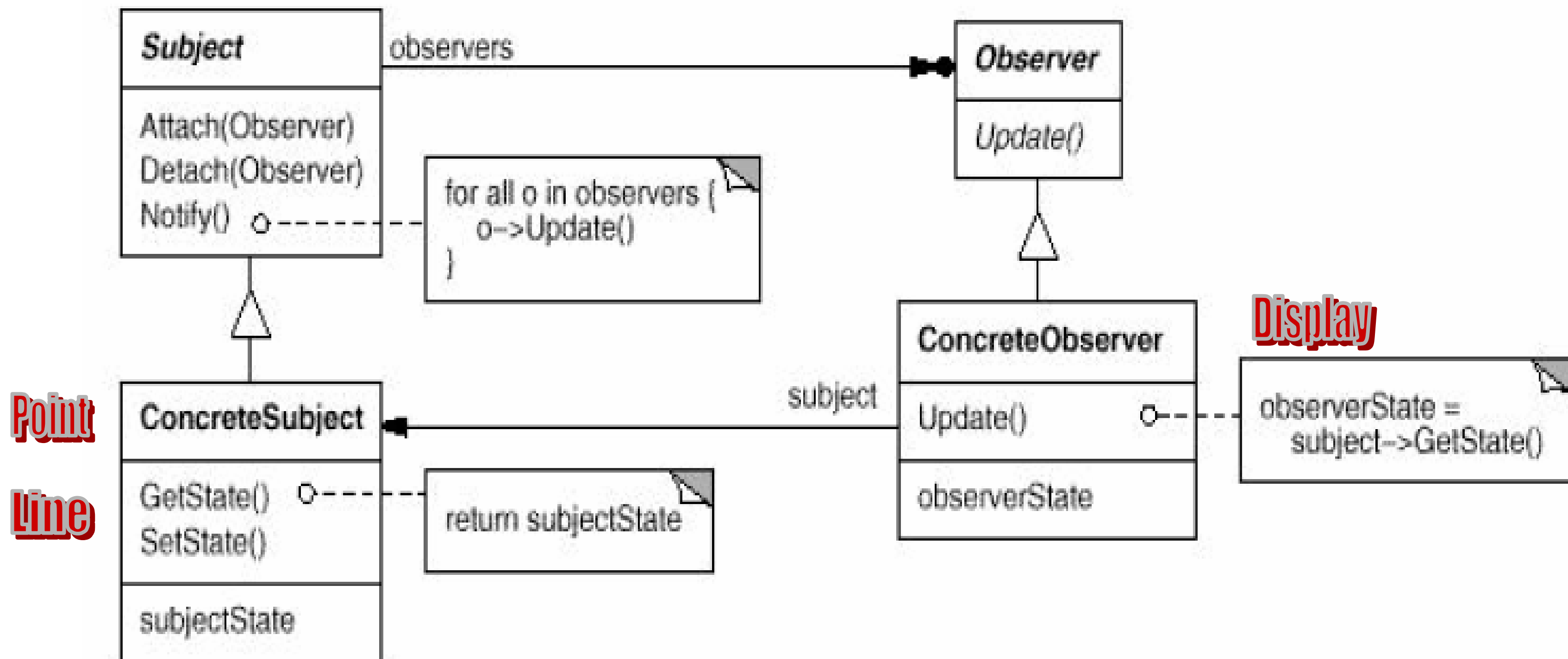



- Extensão através de Agregação




- Modificação não-invasiva
  - chamadas ao método **update()** espalhadas

# Example
## Design patterns



# Exemplo

## ↳Editor de Figuras

- Problema II: Rastreamento
  - mostrar os nomes dos métodos executados antes e após a consulta/ modificação de um elemento
    (Figure, Line ou Point).

- Identificação
  - métodos que consultam ou modificam elementos
    int Point.getX(int), int Point.getY(), void Point.setX(int),
    void Point.setY(int), Point Line.getP1(), Point Line.getP2(),
    void Line.setP1(Point), void Line.setP2(Point)
  - métodos que mostram nomes dos métodos
    **Rastreamento.entry(String)**, **Rastreamento.exit(String)**

- Solução
  **?**

# Exemplo

## Editor de Figuras

**(Rastreamento)**

```
class Rastreamento {
  public static void entry(String s) {
     System.out.println("entry: " + s);
  }

  public static void exit(String s) {
     System.out.println("exit: " + s);
  }
}
```

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

  Point getX() { … return x; … }
  Point getY() { … return y; … }

  void setX(int x) {
    Rastreamento.entry(“setX”);
    this.x = x;
    Rastreamento.exit(“setX”);
  }
  void setY(int y) {
    Rastreamento.entry(“setX”);
    this.y = y;
    Rastreamento.exit(“setY”);
  }
}
```

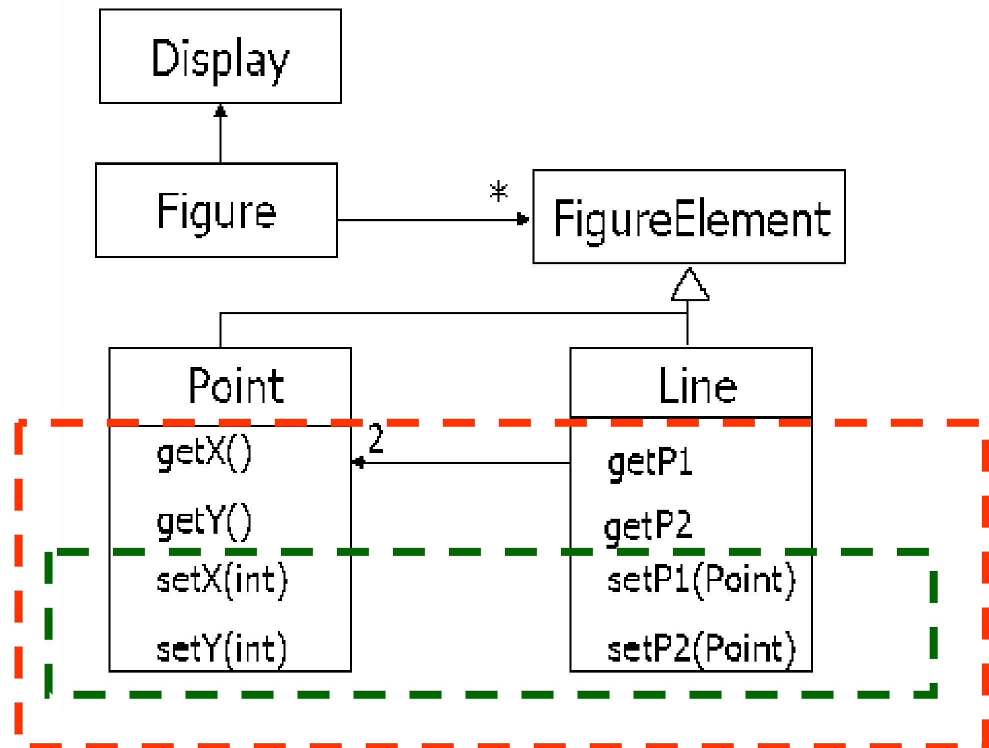
- chamadas aos métodos **entry()** e **exit()** estão espalhadas e misturadas no código

# Exemplo

## Editor de Figuras

- coesão baixa
- acoplamento alto

```
class Point {
  private int x=0, y=0;
…
 public int getX() {
    Rastreamento.entry("getX");
    return x;
    Rastreamento.exit("getX");
  }
 public void setX(int x) {
    Rastreamento.entry("setX");
    this.x = x;
    Display.update();
    Rastreamento.exit("setX");
 }
…
}
```

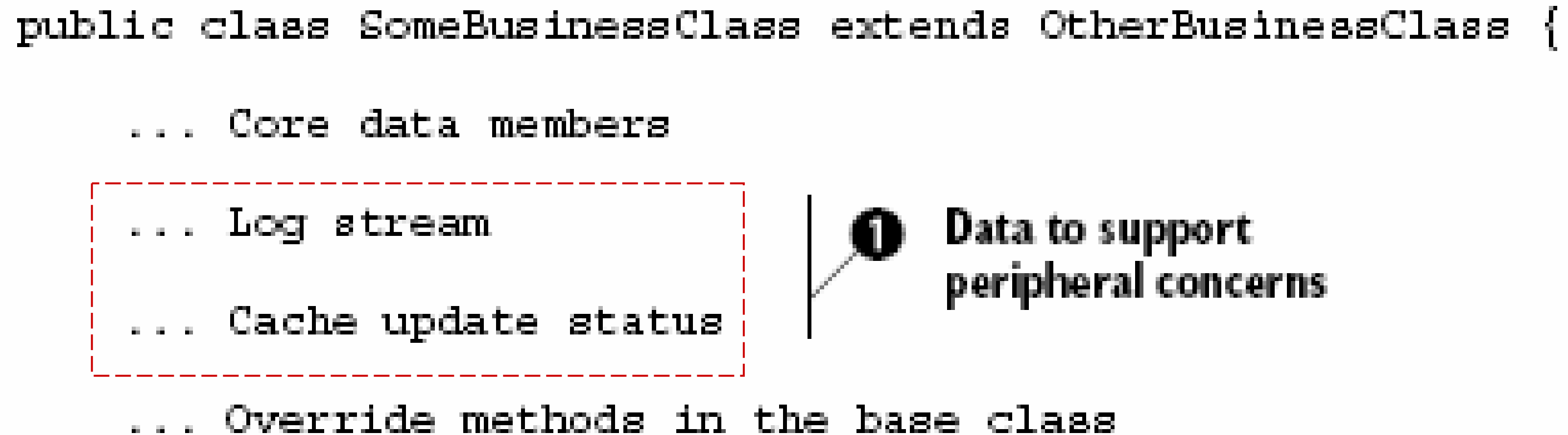


# Outro Exemplo

```
public class SomeBusinessClass extends OtherBusinessClass {
    ... Core data members

    ... Override methods in the base class

    public void someOperation1(<operation parameters>,
                                ...) {

        ... Perform the core operation

    }

    ... More operations similar to above

}
```

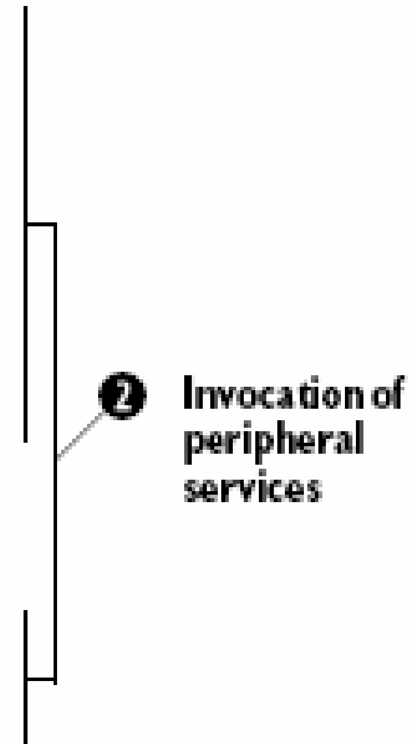
Laddad, R., AspectJ in Action: Practical Aspect-Oriented Programming, Manning Publications Company, 2003.

# Dados para dar suporte a interesses periféricos



# Chamada a serviços periféricos

```
public void someOperation1(<operation parameters>,
                           <authenticated user>,
                           ...) {
   ... Ensure authorization

   ... Ensure info satisfies contracts

   ... Lock the object to ensure thread-safety

   ... Ensure cache is up-to-date

   ... Log the start of operation

   ... Perform the core operation

   ... Log the completion of operation

   ... Unlock the object
```



# Serviços periféricos

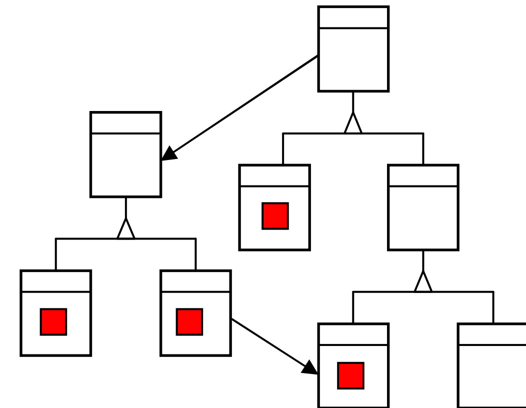
```
    ... More operations similar to above addressing multiple concerns

    public void save(<persitance storage parameters>) {
        ...
    }

    public void load(<persitance storage parameters>) {
        ...
    }
}
```

❸ Methods to support peripheral services

# Limitações de OO

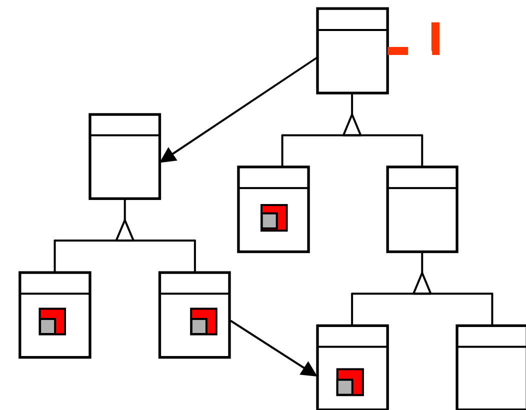
## Sintomas

- espalhamento *(scattering)*
  - fenômeno no qual a implementação de uma ou mais propriedades espalha-se por diversos componentes do sistema



```
Rastreamento.entry("setX("+x+")");
this.x = x;
Rastreamento.exit("setX()");
```

- entrelaçamento *(tangling)*
  - fenômeno no qual a implementação de uma ou mais propriedades mistura-se à implementação de um ou mais componentes do sistema



# Limitações de OO

## ↳Problemas

- Redundância
  - muitos fragmentos de código semelhantes em diversos pontos do código-fonte
- Fraca coesão
  - métodos das classes afetadas contêm instruções que não estão diretamente relacionadas à funcionalidade que implementam
- Forte acoplamento
  - métodos das classes afetadas precisam conhecer métodos das classes que implementam funcionalidade espalhada.

# Limitações de OO

## Consequências

- Dificuldade de compreensão
- Dificuldade de manutenção
- Dificuldade de evolução
- Dificuldade de reutilização

**E os benefícios da orientação a objetos ?...**

# Diagnosticando o Problema

## Definição

- Se a funcionalidade R
  - está misturada a outras funcionalidades
  - está espalhada pela hierarquia de classes

- Então dizemos que a funcionalidade R
  - entrecorta uma ou mais classes
    - *crosscuts*

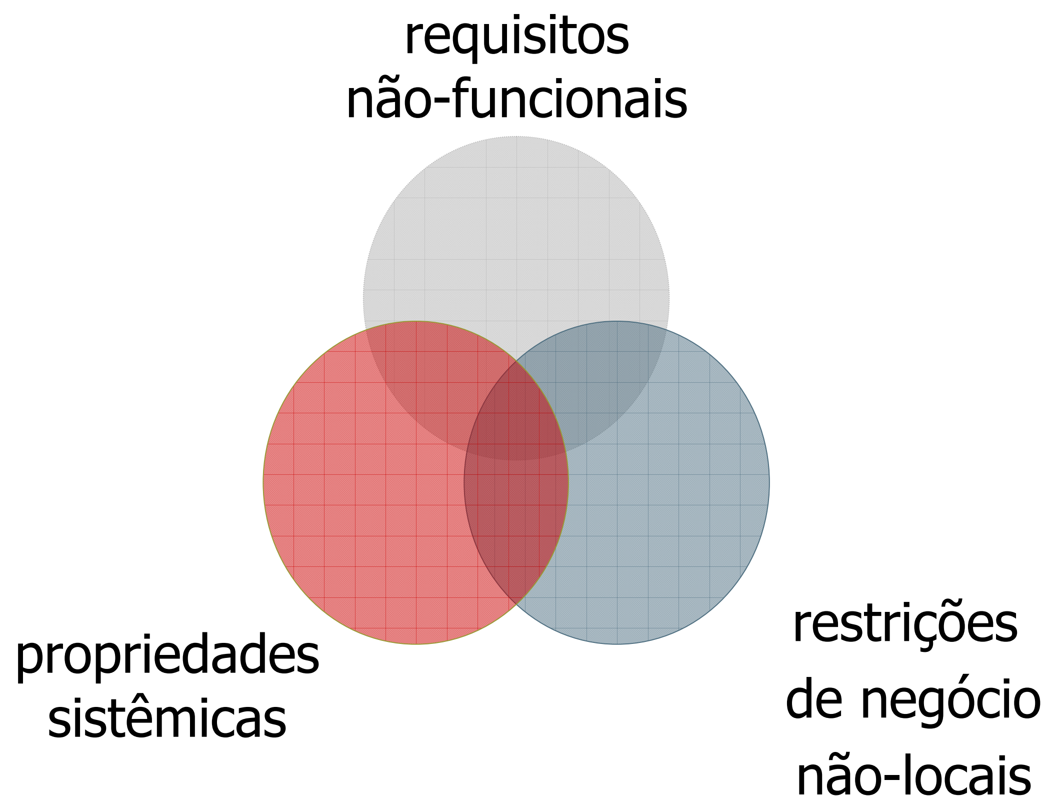
R é um interesse transversal

# Definição

Crosscutting concern

- Definições
  - *concern*
    - interesse
  - *crosscut*
    - *cut across* = cortar através
    - entrecortar
  - *crosscutting concern*
    - interesse transversal

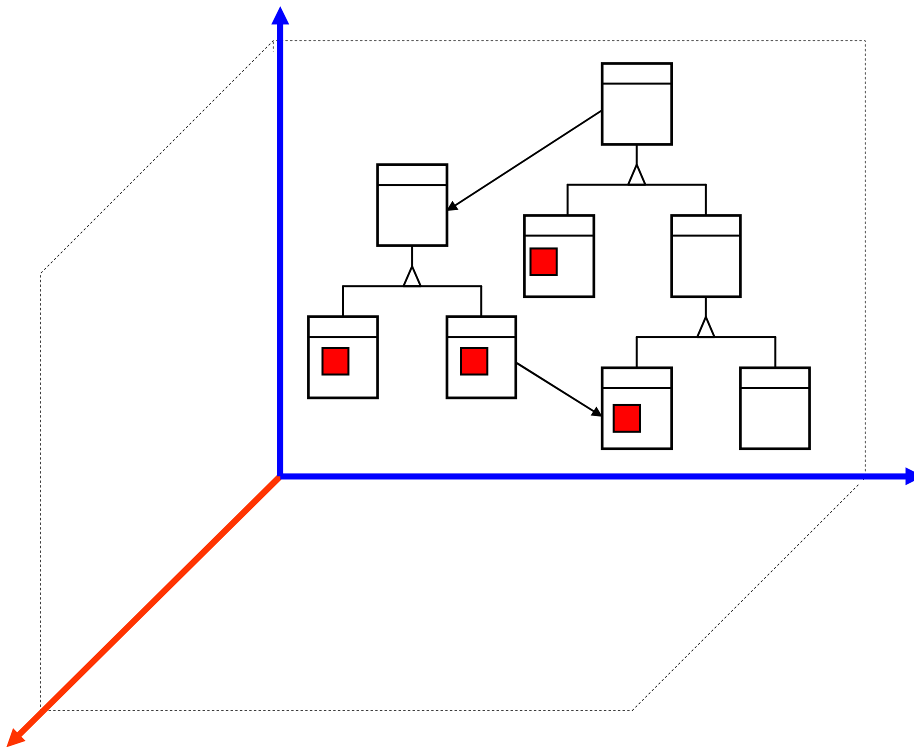
# Interesses transversais




Persistência
Distribuição
Controle de concorrência
Tratamento de erros
Rastreamento (*Tracing*)
Depuração
Auditoria (*Logging*)
Contratos
…
…

# Limitações de OO

## ↳Análise do Problema

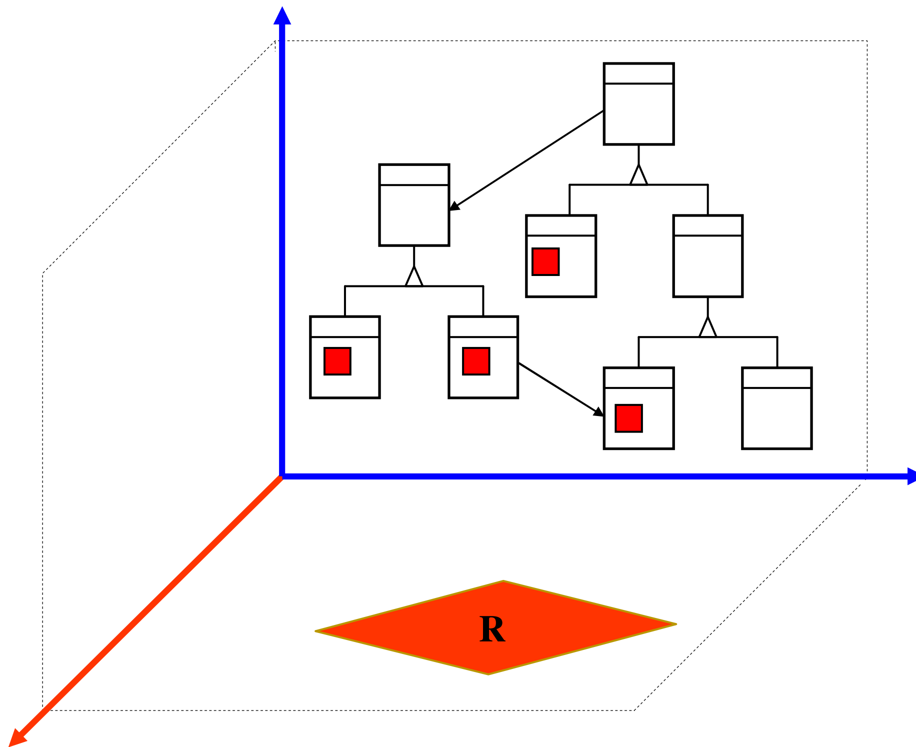


- Apesar dos problemas apresentados, OO cumpre bem muitas de suas promessas…
- …e algumas propostas tentam solucionar estes problemas sem perder de vista os benefícios da OO

■ **Interesse transversal "R"**

# Interesse Transversal "R"

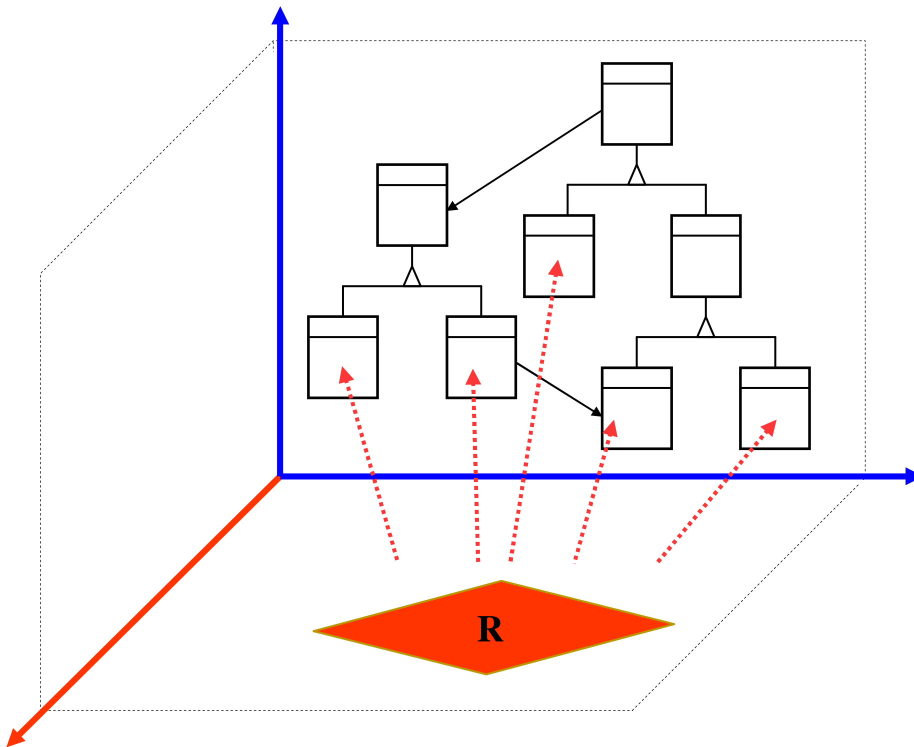
## Uma Solução




- Separar (isolar)
- Modularizar
  - Novo tipo de módulo

# Interesse Transversal "R"

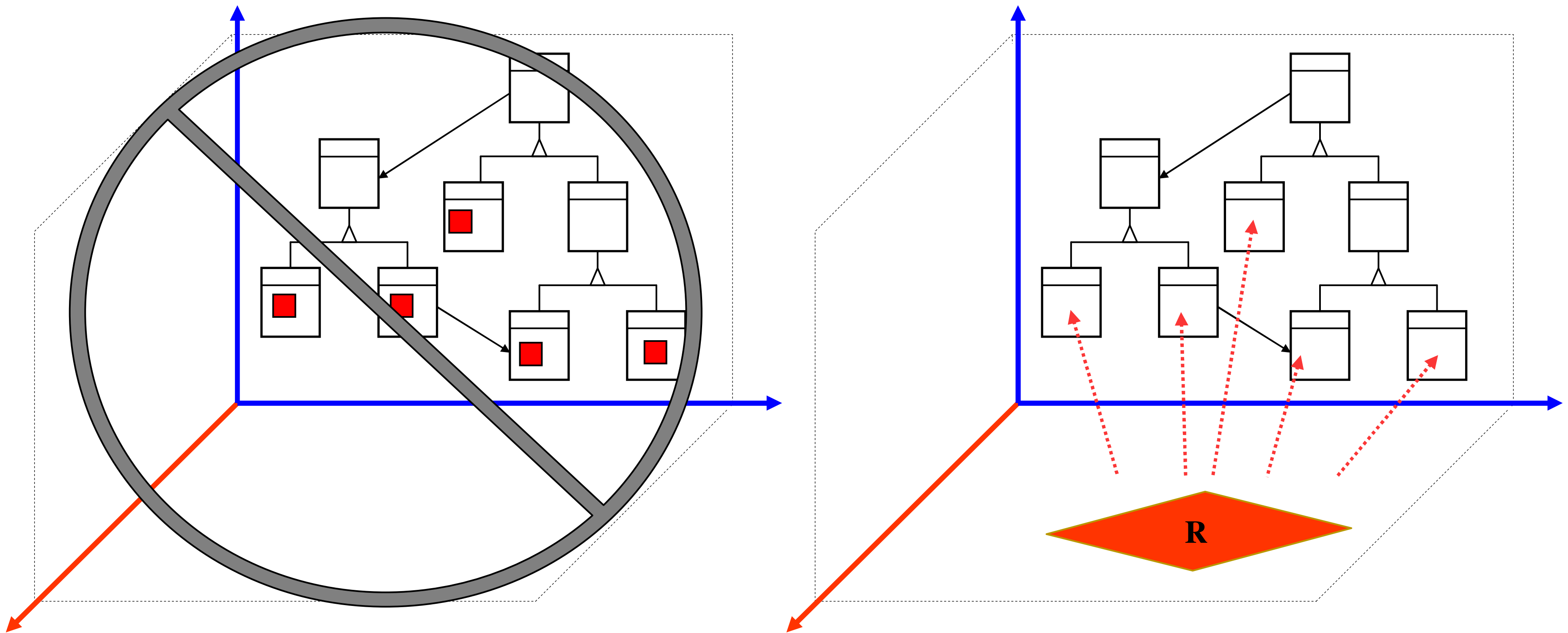
## ↳Uma Solução




- Compor
  - combinar nos locais desejados
  - novo mecanismo de composição!
- Reutilizar
  - combinar em outros locais

# Interesse Transversal "R"

## ↳Uma Solução com Aspectos (POA)



# Disque Saúde com POA

```
public class Complaint {
    private String description;
    private Person complainer; ...
    public Complaint(String description, Person complainer, ...) {
        ...
    }
    public String getDescription() {
        return this.description;
    }
    public Person getComplainer() {
        return this.complainer;
    }
    public void setDescription(String desc) {
        this.description = desc;
    }
    public void setComplainer(Person complainer) {
        this.complainer = complainer;
    }
}
```

```
public class Person {
    private String nome; ...
    public Person(String nome, ...) {
        this.matricula = matricula; ...
    }
    public String getNome() {
        return nome;
    } ...
}
```

```
public class HealthWatcherFacade {
    public void update(Complaint complaint)
            throws TransactionException, RepositoryException,
            ObjectNotFoundException,
ObjectNotValidException {
        ...
    }
}
```

```
public class ServletUpdateComplaintData extends HttpServlet {
    private HealthWatcherFacade facade;
    public void init(ServletConfig config) throws ServletException {
        try {
            facade = HealthWatcherFacade.getInstance();
        }
        catch (Exception ex) {...}
    }
    public void doPost(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        ...
    } ...
}
```

Sistema local

Aspectos de Distribuição para RMI

```
public interface IFacade extends java.rmi.Remote {

    public void updateComplaint complaint)
            throws TransactionException, RepositoryException,
            ObjectNotFoundException, ObjectNotValidException,
            RemoteException;
    . . .
}
```

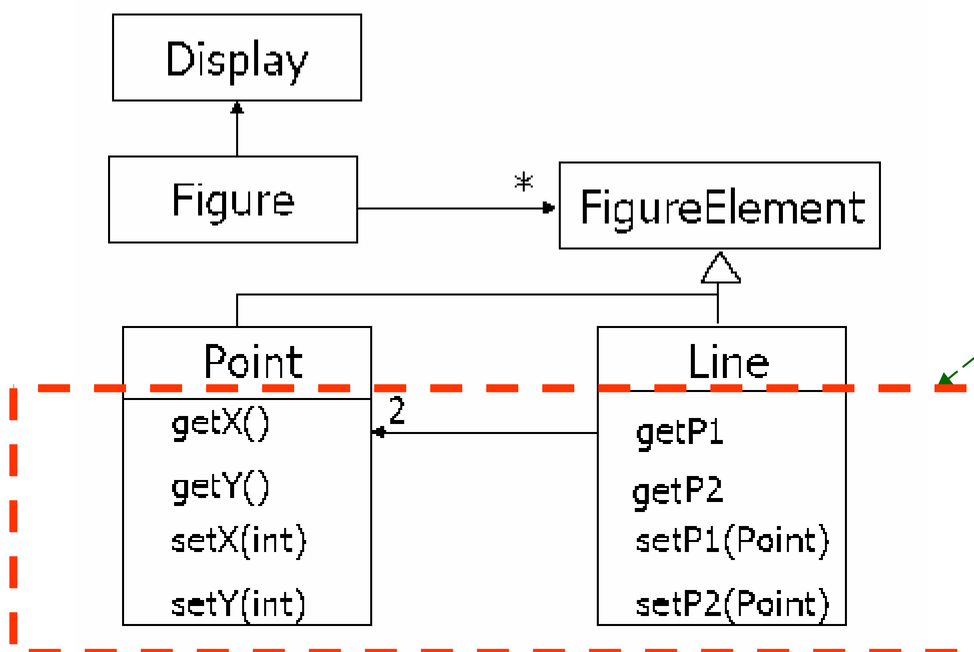
```
aspect DistributionAspect {
    declare parents: HealthWatcherFacade implements IFacade;
    declare parents: Complaint || Person implements java.io.Serializable;
    public static void HealthWatcherFacade.main(String[] args) {
        try {
            HealthWatcherFacade facade = HealthWatcherFacade.getInstance();
            System.out.println("Creating RMI server...");
            UnicastRemoteObject.exportObject(facade);
            java.rmi.Naming.rebind("/HealthWatcher");
            System.out.println("Server created and ready.");
        } catch (RemoteException rmiEx) {...}
        catch (MalformedURLException rmiEx) {...}
        catch(Exception ex) {...}
    }
    private IFacade remoteFacade;
    pointcut facadeMethodsExecution():
        within(HttpServlet+) && execution(* HealthWatcherFacade.*(..)) &&
        this(HealthWatcherFacade);
    before(): facadeMethodsExecution() { prepareFacade();}
    private synchronized void prepareFacade() {
        if (healthWatcher == null) {
            try { remoteFacade = (IFacade) java.rmi.Naming.lookup("//HealthWatcher");
            } catch (java.rmi.RemoteException rmiEx) {...}
            catch (java.rmi.NotBoundException rmiEx) {...}
            catch (java.net.MalformedURLException rmiEx) {...}
    }
    void around(Complaint complaint) throws TransactionException, RepositoryException
            ObjectNotFoundException,ObjectNotValidException:
            facadeRemoteExecutions() && args(complaint) &&
            call(void update(Complaint)) {
        try { remoteFacade.update(complaint);
        } catch (RemoteException rmiEx) {...}
    }
}
```

S.Soares, E. Laureano, P. Borba. "**Implementing Distribution and Persistence Aspects with AspectJ**". Proceedings of OOPSLA'02, November 2002.

# Exemplo

## Editor de Figuras com POA

- Solução detalhada na Parte II




Rastreamento

# Exemplo

## Editor de Figuras

**(Rastreamento)**

```
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; }
}
```

```
aspecto Rastreamento {

 public static void entry(String s) {
    System.out.println("entry: " + s);
 }

 public static void exit(String s) {
    System.out.println("exit: " + s);
 }

  para toda chamada C de método que consulta
  ou modifica elemento
```

chamadas para:
int Point.getX(int), int Point.getY(), void Point.setX(int),
void Point.setY(int), Point Line.getP1(), Point Line.getP2(),
void Line.setP1(Point), void Line.setP2(Point)

```
  ANTES da chamada C,
        execute entry(mensagem)
  APÓS a chamada C,
        execute exit(mensagem)
}
```

# Exemplo
## Editor de Figuras

(Rastreamento)

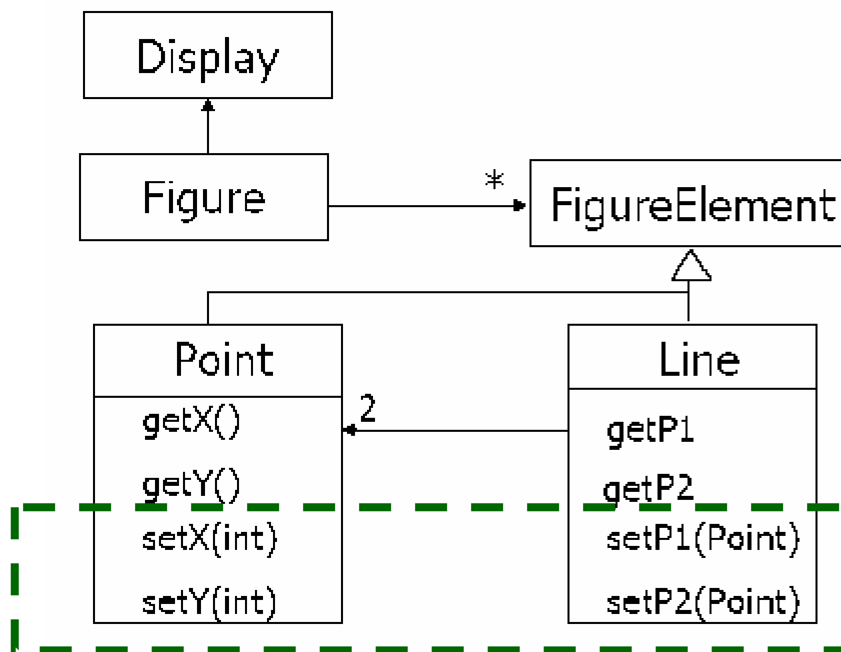
```
class Line {
  private Point p1, p2;

  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 Rastreamento {

 public static void entry(String s) {
    System.out.println("entry: " + s);
 }

 public static void exit(String s) {
    System.out.println("exit: " + s);
 }

 pointcut traceCalls():
   (call(* Point.set*(int))  ||
    call(* Line.set*(Point)) ||
    call(* Point.get*())     ||
    call(* Line.get*()));

  before(): traceCalls () {
     entry("method " + thisJoinPoint);
  }
  after() returning: traceCalls() {
     exit("method " + thisJoinPoint);
  }
}
```

# Exemplo

## Editor de Figuras com POA

- Solução detalhada na Parte II



# Exemplo

## Editor de Figuras

**(DisplayUpdating)**

```
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; }
}
```

```
aspecto DisplayUpdating {

  para toda chamada de método C que
  modifica elemento

  chamadas para:
  void Point.setX(int),
  void Point.setY(int),
  void Line.setP1(Point),
  void Line.setP2(Point)

   APÓS chamada de método C,
       execute Display.update()
}
```

# Exemplo

## Editor de Figuras

**(DisplayUpdating)**

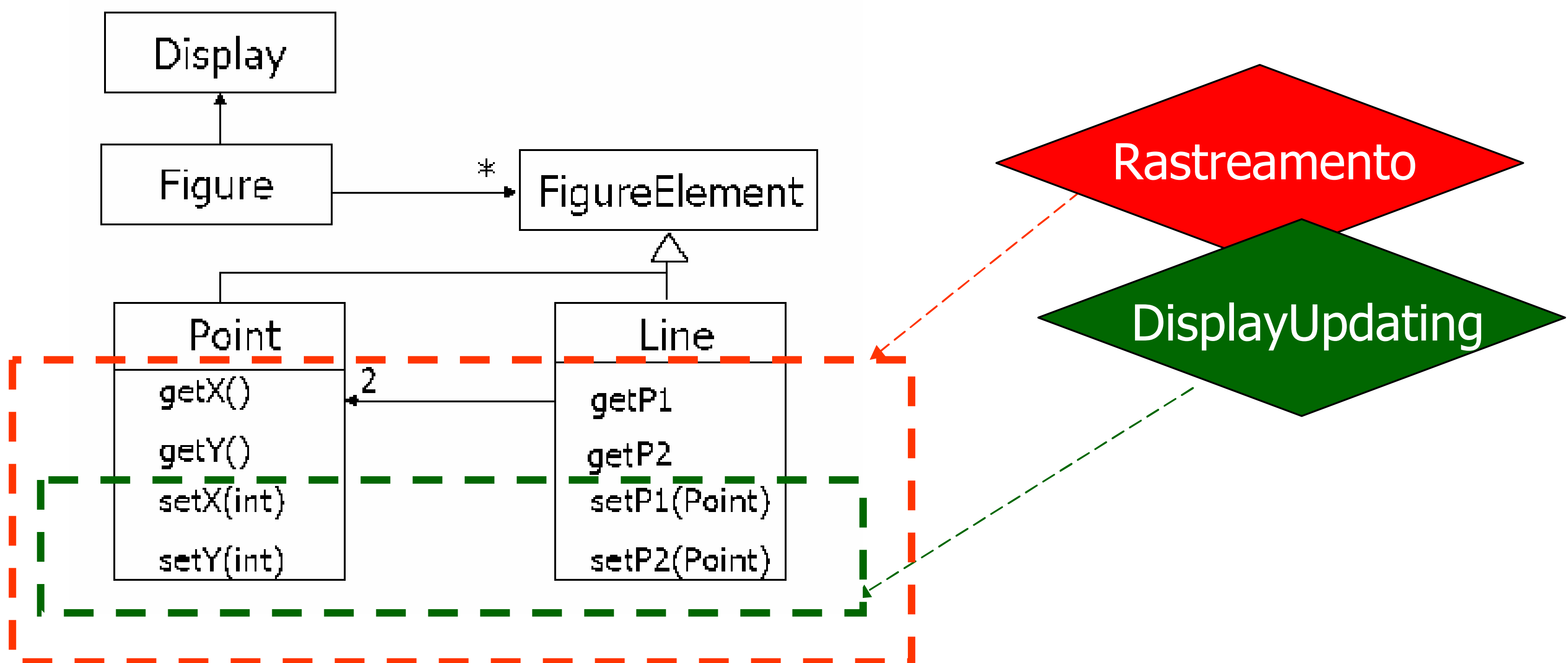
```
class Line {
  private Point p1, p2;

  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 Line.setP1(Point))     ||
      call(void Line.setP2(Point))     ||
      call(void Point.setX(int))       ||
      call(void Point.setY(int)));

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

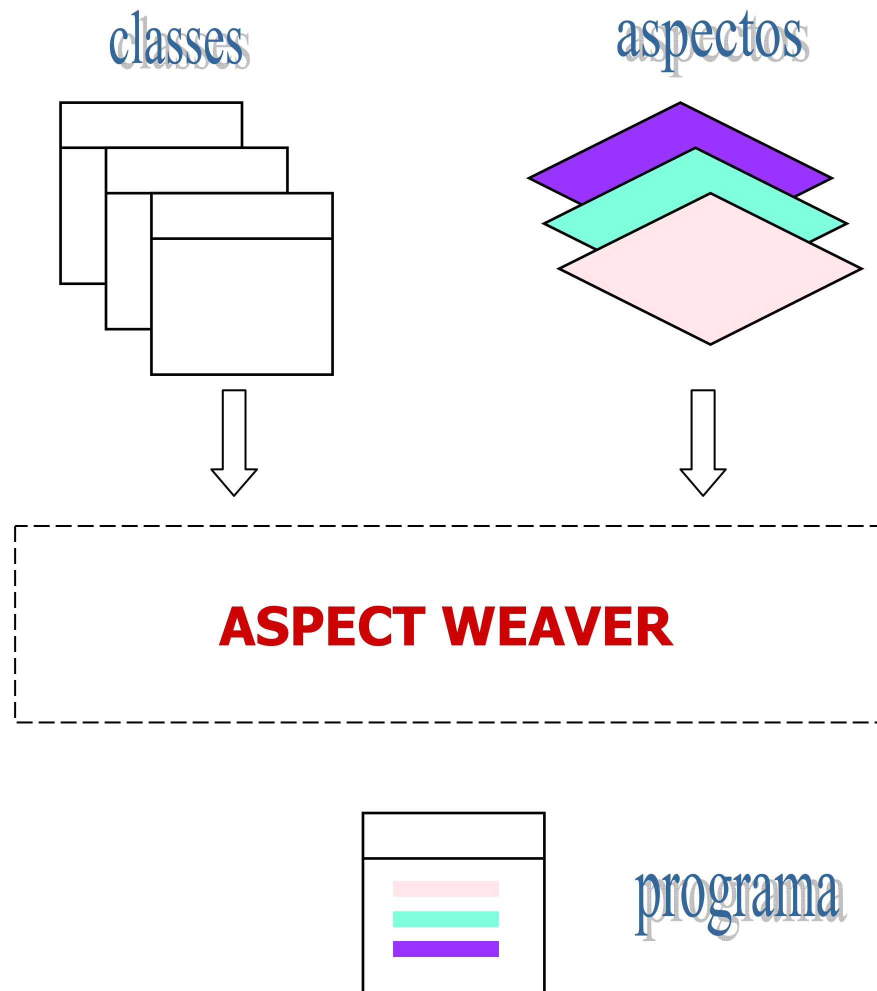
# Exemplo

## Editor de Figuras com POA

- Solução detalhada na Parte II



# Combinação



# Exemplo: Antes da combinação

## Uma Classe: SomeBusinessClass

```
public class SomeBusinessClass extends OtherBusinessClass {
    ... Core data members

    ... Override methods in the base class

    public void someOperation1(<operation parameters>,
                                    ...) {

        ... Perform the core operation

    }

    ... More operations similar to above

}
```

Laddad, R., AspectJ in Action: Practical Aspect-Oriented Programming, Manning Publications Company, 2003.

# Exemplo: Antes da combinação

## Um Aspecto: Logging

- Descrição em linguagem natural:
  1. Crie um objeto para "registro de operações"
  2. Registre o início de toda operação pública
  3. Registre o final de toda operação pública

- Considere a interface:

```
public interface Logger {
    public void log(String message);
}
```

# Exemplo: Após a combinação entre a classe e o aspecto

```
public class SomeBusinessClass extends OtherBusinessClass {
    ... Core data members

    ... Override methods in the base class

  Logger _logger = ...

    public void someOperation1(<operation parameters>,
                                                  ...) {
      _logger.log("Starting someOperation1");

        ... Perform the core operation

      _logger.log("Completed someOperation1");
    }

    ... More operations similar to above
    ... Each public operation will be similarly
    ... woven in with log statements

}
```

Rule 1—creating a logger object

Woven in automatically

Rule 2—logging the beginning of the operation

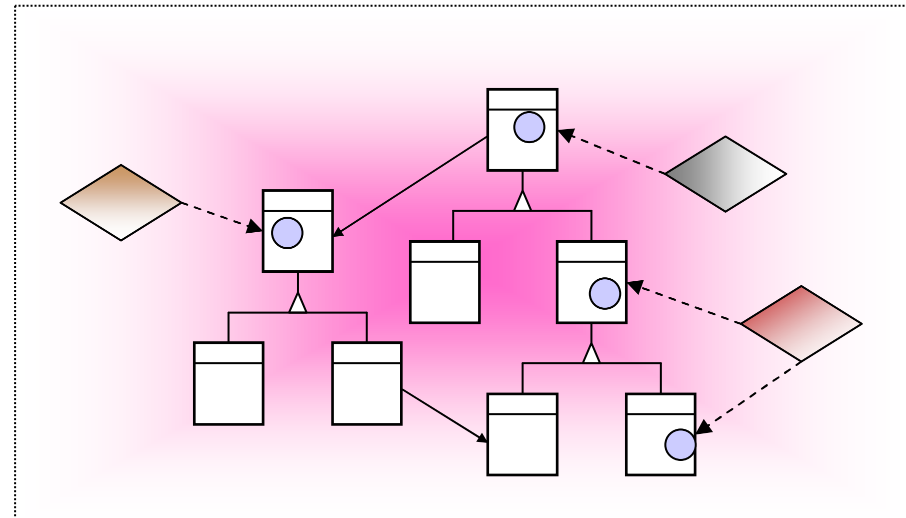
Woven in automatically

Rule 3—logging the completion of the operation

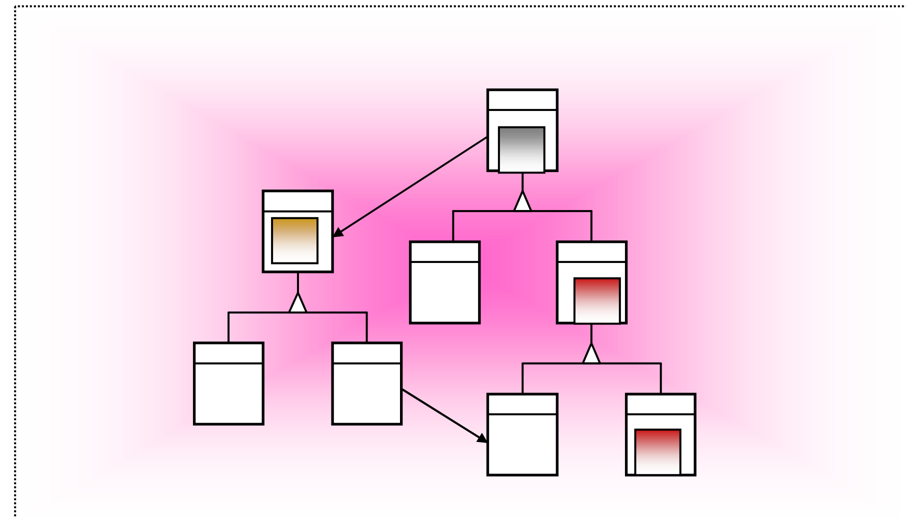
Woven in automatically

# Modelo de aspectos

- Arcabouço Conceitual
  - componente
  - ponto de junção
  - aspecto
    - especificação de pj
    - especificação de ação
  - crosscutting



  - combinação (weaving)
    - estática
    - dinâmica



# Modelo de aspectos

- Ponto de junção (*join point*)
  - ponto relativo ao componente (classe, objeto, etc.) que será afetado pelo aspecto

- Variações
  - pontos de junção dinâmicos
    - chamada de método
    - levantamento de exceção

  - pontos de junção estáticos

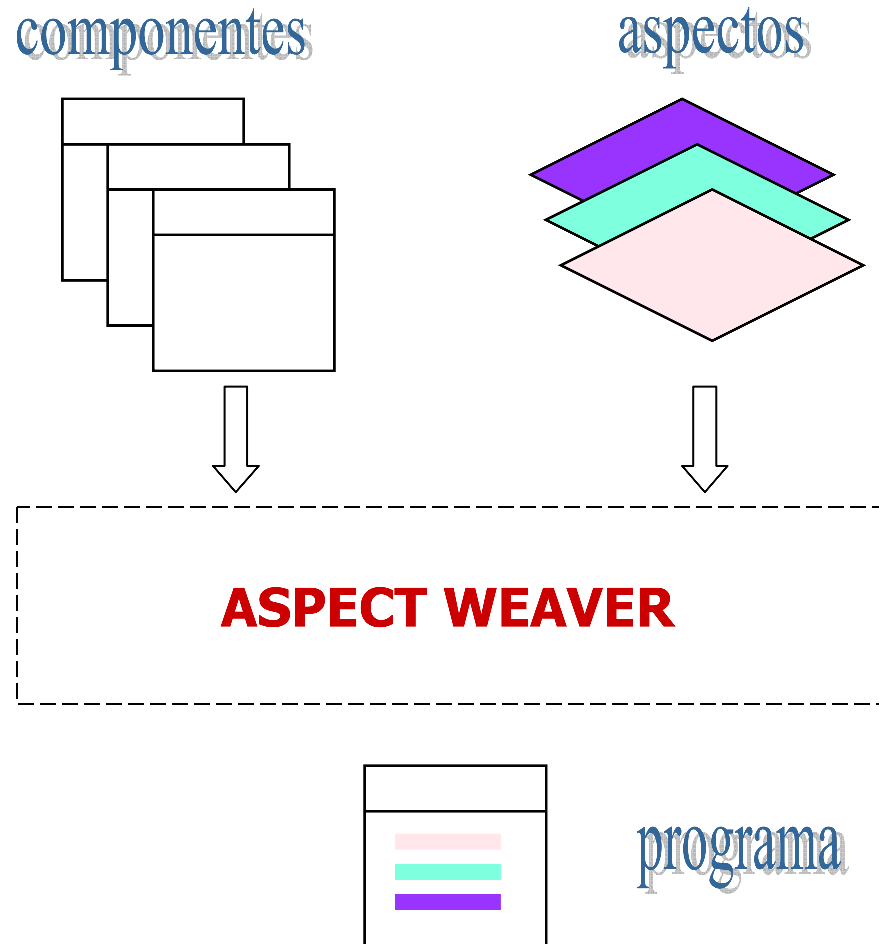
# Modelo de aspectos

- aspecto
  - especificação de pontos de junção
    pontos de corte (pointcuts)
  - especificação de ação
    adendo (advice)

**para toda chamada de método C que modifica elemento**

**APÓS chamada de método C,**
**execute Display.update()**

# Modelo de Aspectos
# Combinação Estática (*Static Weaving*)

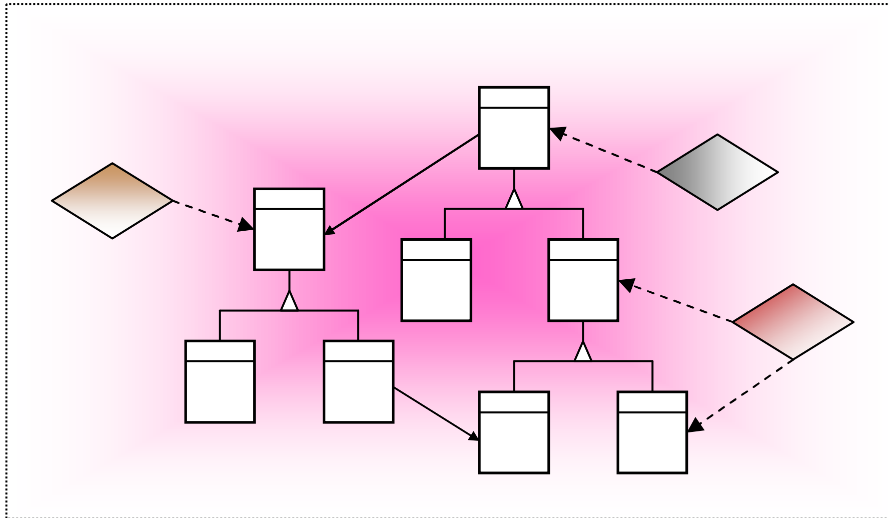


# Modelo de aspectos

- Princípios
  - Separação de interesses
    - suporte a separação de interesses transversais
  - Modularidade
    - aspectos modularizam interesses transversais

- Propriedades
  - Dicotomia aspecto-base
    - aspecto ≠ componente (classes, etc.)
  - Quantificação (*Quantification*)
    - aspectos entrecortam um número arbitrário de componentes; indicam pontos de junção
  - Inconsciência (*Obliviousness*)
    - componentes não precisam conhecer nem estar preparados para aspectos
  - Inversão de dependência

# Propriedades



- Inversão de dependência
- Quantificação
  *para todo ... faça ...*
- Inconsciência
- Dicotomia aspecto-base

R. Filman and D. Friedman. **Aspect-oriented programming is quantication and obliviousness**. In OOPSLA 2000 Workshop on Advanced Separation of Concerns, Minneapolis, MN, Oct. 2000.

# Programação Orientada a Aspectos (POA)

- orientação a aspectos
  - o que é um aspecto?
  - para que serve?
  - quais os benefícios?
- novo paradigma
  - aspectos substituem objetos?
- novo tipo de decomposição
  - decomposição orientada a aspectos?
- linguagens, ferramentas?
- aplicações
- desafios?
- identificação, separação, representação e composição de *aspectos* em várias atividades do processo de software

# 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.
- Ênfase no uso eficiente de <u>mecanismos de linguagem</u> para concretizar os conceitos e propriedades do modelo de aspectos no <u>nível de programação</u>

```
aspect Caller
perthis(this(Information_Agent||SearchResultReceivi
ngPlan)) {

public String Information_Agent.answerer;

  // after sendAndLockInformationAsk,
sendMsg(msg) and synchronizes
  public void
Information_Agent.sendAndLockInformationAsk(Goa
lMsg msg) {
...
 //*********caller-related points that crosscuts the
agent`s state and behavior*********//

  pointcut activatesCaller(Information_Agent agent,
String keyword):
     (args(keyword) && this(agent) &&
execution(String search(String)));

  after(Information_Agent agent, String keyword):
activatesCaller(agent, keyword) {
    ...
  }

pointcut
receiveSearchResult(SearchResultReceivingPlan
plan, PAgent agent, Goal goal):
     (args(agent,goal) && this(plan) && execution(void
executePlan(PAgent,Goal)));

  after(SearchResultReceivingPlan plan, PAgent
agent, Goal goal):
          receiveSearchResult(plan, agent, goal) {
...
  }
}
```

```
aspect Answerer
perthis(this(Information_Agent||SearchAskAnswering
Plan)) {

  //****answerer interface introduced to the
agent***//

       public String Information_Agent.caller;
       public String
Information_Agent.doSearchAsk(String keyword) {
...
       }

  // after sendSearchResult, sendMsg(msg)
       public void
Information_Agent.sendSearchResult(GoalMsg msg)
       { ...   }

  //*********answerer-related points that crosscuts
the agent`s state and behavior*********//

  pointcut
receiveSearchAsk(SearchAskAnsweringPlan plan,
PAgent agent, Goal goal):
     (args(agent, goal) && this(plan) &&
execution(void executePlan(PAgent, Goal)));

  after(SearchAskAnsweringPlan plan, PAgent
agent, Goal goal):
          receiveSearchAsk(plan, agent, goal) {
   ...
  }
}
```

```
aspect Collaboration dominates Interaction
   perthis(this(Information_Agent||User_Agent)) {

  //**** interface introduced to the agent *****//

  public SharedObject PAgent.shared = new
SharedObject();

  public SharedObject PAgent.getSharedObject() {
     return shared;
  }

  public void PAgent.setSharedObject(SharedObject
so) {
     shared = so;
  }
// pointcuts

  pointcut synchronization(PAgent agent):
      (this(agent) && execution(void
sendAndLock*(Message+)));


  pointcut unlock(PAgent agent, Goal goal):
      (args(goal) && this(agent) && execution(void
receiveAndUnlock*(Goal+)));

  after (PAgent agent): synchronization(agent){
    ...
  }

  after (PAgent agent, Object obj): unlock(agent, obj)
{  ...  }
}
```

# o que é um aspecto?

```
aspect Rastreamento {

 public static void entry(String s) {
    System.out.println("entry: " + s);
 }

 public static void exit(String s) {
    System.out.println("exit: " + s);
 }

 pointcut traceCalls():
   (call(* Point.set*(int))   ||
    call(* Line.set*(Point)) ||
    call(* Point.get*())      ||
    call(* Line.get*()));

  before(): traceCalls () {
     entry("method " + thisJoinPoint);
  }
  after() returning: traceCalls() {
     exit("method " + thisJoinPoint);
  }
}
```

- Abstração
- Unidade de modularização
- Um elemento de programação que localiza e separa um interesse transversal

Implementação

# quais os benefícios?

- Modularidade adequada na presença de interesses transversais
- Separação de interesses funcionais de interesses não-funcionais

- Facilidade de compreensão +
- Facilidade de manutenção +
- Facilidade de evolução +
- Facilidade de reutilização +
- Rastreabilidade dos interesses +

# novo paradigma?

- Não
  - Ptolomeu x Copérnico

- Sim
  - Programação imperativa x programação orientada a objetos

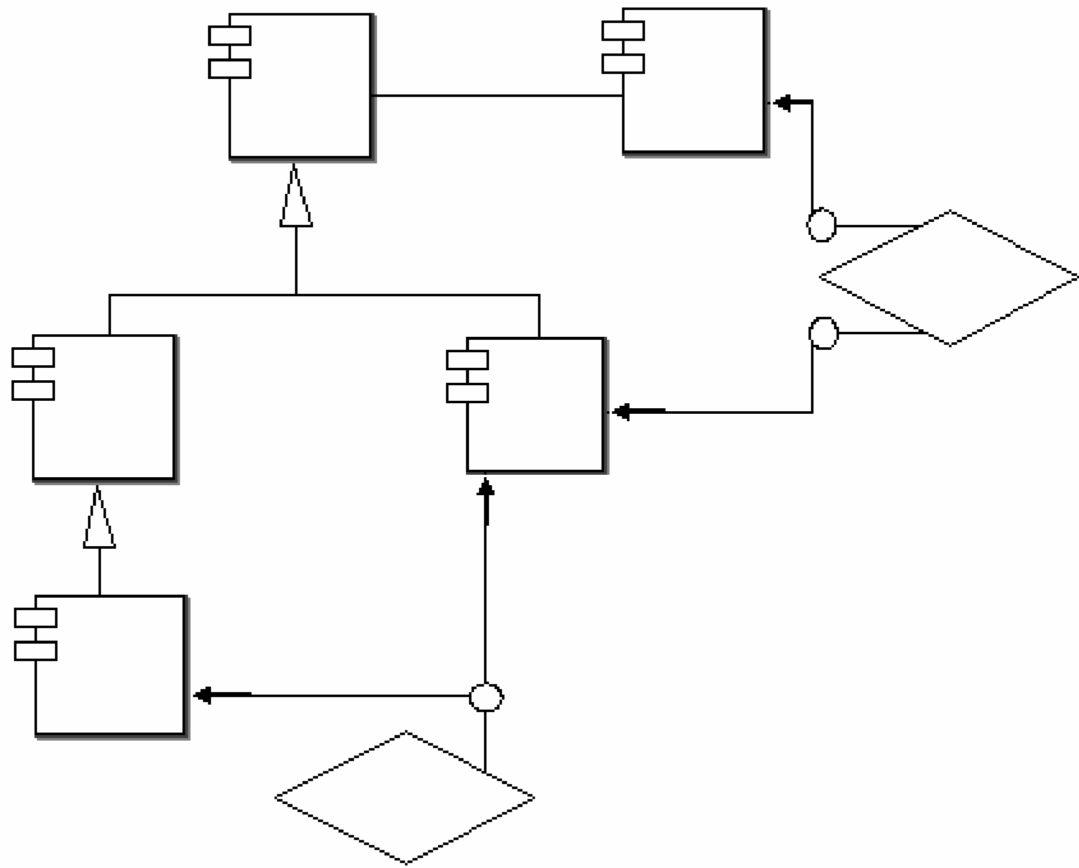
# aspectos substituem objetos?

- Não
  - aspectos ajudam a contornar algumas limitações de objetos
    - modularização de interesses transversais

- Por outro, aspectos não são específicos para uso com objetos

# decomposição orientada a aspectos?

- aspectos e "componentes" (base)
- aspectos afetam componentes



# linguagens, ferramentas?

- AspectJ, Aspect*, ...

- Ambientes de programação
  - Combinador de aspectos
  - Editor orientado por sintaxe
  - Visualizador

- Mineração de aspectos
- Refactoring para aspectos

# linguagem de programação orientada a aspectos

- Definição
  - Linguagem de programação que dá suporte aos conceitos e propriedades do modelo de aspectos

- AspectJ
  - aspect
    - pointcut
    - advice
    - inter-type declaration
  - join point
  - weaving

# aplicações?

- Alguns dos principais usos para aspectos (relatados pela comunidade de usuários que adotou AspectJ):
  - desempenho em tempo de execução: otimização de memória, fusão de laços;
  - depuração e instrumentação: rastreamento, auditoria, teste, monitoramento;
  - garantia de propriedades e verificação: garante que os tipos de um framework são usados de forma adequada, validação de componentes, garantia das melhores práticas de programação;
  - configuração: gerenciamento das especificidades do uso de diferentes plataformas;
  - aspectos operacionais: sincronização, cache, persistência, gerenciamento de transações, segurança e balanceamento de carga;
  - tratamento de falhas: redirecionamento de uma chamada com falha para um outro serviço;
  - construção de programas: herança múltipla, papéis e visões;
  - passagem de parâmetros remotos, questões de configuração, restrições em tempo real, tratamento de falhas, segurança, depuração etc.

# desafios?

- identificação, separação, representação e composição de *aspectos* em várias atividades do processo de software
  - identificar candidatos a aspectos
  - identificar pontos de junção entre aspectos e classes
  - compor aspectos
    - múltiplos aspectos
      - precedência
      - interferência
  - outros mais que veremos adiante…

## desenvolvimento de software orientado a aspectos

- identificação, separação, representação e composição de *aspectos* em várias atividades do processo de software

  - Decomposição aspectual
    - Identificação de interesses
  - Implementação de interesses
    - Use objetos para interesses funcionais
    - Use aspectos para interesses não-funcionais
  - Recomposição aspectual
    - Combinação de interesses

# Referências

- **POA**
  - G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J. Loingtier and J. Irwin. “Aspect-Oriented Programming”. June 1997.
  - R. Filman and D. Friedman. Aspect-oriented programming is quantication and obliviousness. In OOPSLA 2000 Workshop on Advanced Separation of Concerns, Minneapolis, MN, Oct. 2000.
  - Elrad, T.; Aksit, M.; Kiczales, G.; Lieberherr, K.; Ossher, H.: “Discussing Aspects of AOP”. Communications of the ACM 44 (10), pp. 33 – 38, October 2001.
  - C. Lopes. “Aspect-Oriented Programming: An Historical Perspective (What’s in a Name?)”. ISR Technical Report #UCI-ISR-02-5, University of California, Irvine, December 2002.
- **AspectJ**
  - G. Kiczales et al. “An Overview of AspectJ”. ECOOP’2001, Budapest, Hungary, 2001.
  - Kiczales, G., Hilsdale, E., Hugunin, J., Kersten, M., Palm, J., Griswold, W. “Getting Started with AspectJ”. Communication of the ACM. October 2001.

# Demonstração Eclipse e AJDT

Demo

# Final da Parte I