IME - USP

AspectJ

Silvio do Lago Pereira

Doutorando em Ciência da Computação

slago@ime.usp.br

Sumário

- Programação Orientada a Objetos (POO)
- Programação Orientada a Aspectos (POA)
- AspectJ
- Exemplos

IME - USP

Programação Orientada a Objetos

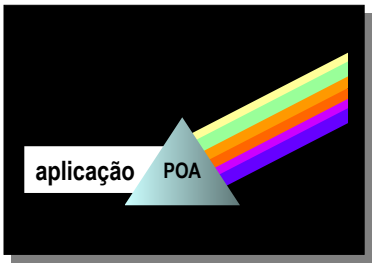
Limitações da POO
Requisitos transversais
Rastreamento usando POO
Problemas com essa abordagem
A causa dos problemas da POO

Limitações da POO

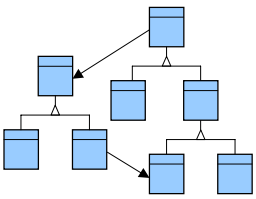
- POO é o paradigma de programação da atualidade:
  - Engenharia de Software
  - metodologias e ferramentas
- POO tem algumas limitações:
  - requisitos transversais (*cross-cutting concerns*)
  - encapsulamento de requisitos transversais

Exemplos de requisitos transversais

- segurança
- persistência
- auditoria e depuração
- tratamento de exceções
- otimização de desempenho
- concorrência e sincronização
- regras e restrições do negócio

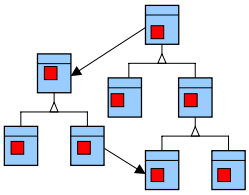


Rastreamento usando POO



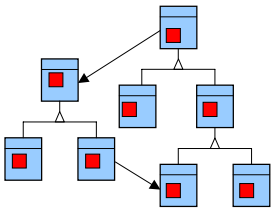
```
class Tracing {
  public static void entry(String s) {
    System.out.println("entry: " + s);
  }
  public static void exit(String s) {
    System.out.println("exit: " + s);
  }
}
```

```
class Person {
  private String name = "";
  public void setName(String name) {
    Tracing.entry("setName("+name+")");
    this.name = name;
    Tracing.exit("setName()");
  }
  ...
}
```

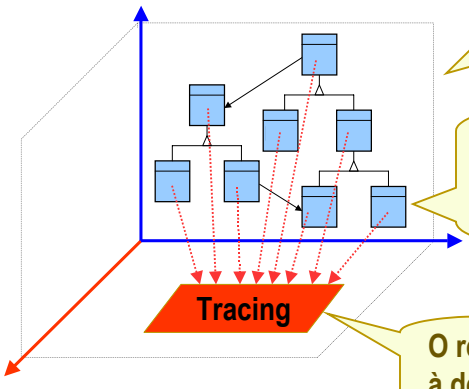


Problemas com essa abordagem

- Redundância
- Fraca coesão
- Forte acoplamento
- Dificuldade de compreensão
- Dificuldade de manutenção
- Dificuldade de reutilização



A causa dos problemas da POO



A decomposição primária na POO é feita no plano.

O requisito transversal fica espalhado pelas classes porque é tratado na dimensão errada!

O requisito transversal é ortogonal à decomposição primária!

IME - USP

Programação Orientada a Aspectos

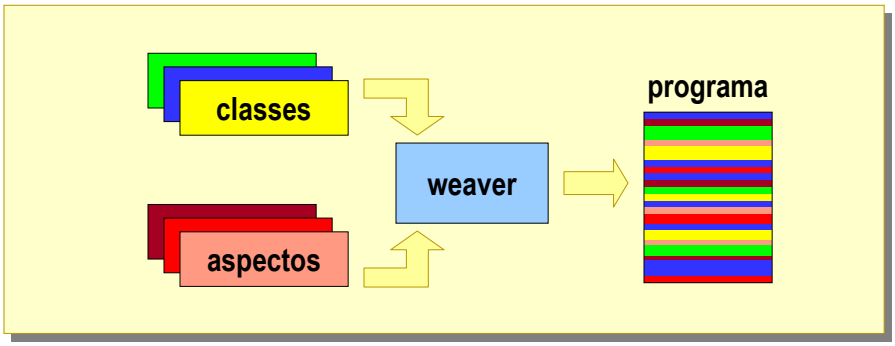
Princípios básicos da POA
Ingredientes essenciais da POA
Benefício esperado da POA

Princípios básicos da POA

- A separação de requisitos transversais é um princípio que deve guiar todas as etapas numa metodologia de desenvolvimento de software.
- A modularização de requisitos transversais deve ser feita através de uma nova unidade de encapsulamento, denominada **aspecto**.

Ingredientes essenciais na POA

- **Classes**: encapsulam requisitos funcionais.
- **Aspectos**: encapsulam requisitos transversais.
- **Weaver**: entrelaça classes e aspectos num programa.



Benefício esperado da POA

- Boa modularidade, mesmo quando temos requisitos transversais:
  - facilidade de desenvolvimento
  - facilidade de manutenção
  - facilidade de reutilização

IME - USP

AspectJ

O que é AspectJ ?
Aspectos versus classes
Joinpoints e pointcuts
Advices e Introductions
Reusabilidade

O que é AspectJ ?

- É uma extensão de Java, orientada a aspectos.
- É um weaver que implementa POA em Java.
- É um preprocessador.
- Fácil de aprender e utilizar.
- Disponível gratuitamente.

Aspectos versus classes

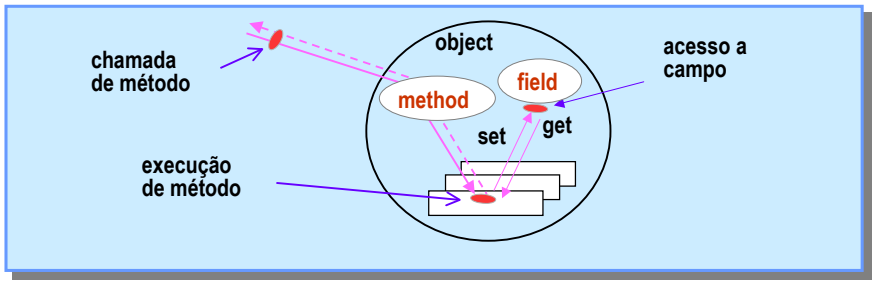
- Aspectos são similares a classes:
  - têm um tipo;
  - podem ser estendidos;
  - podem ser abstratos ou concretos;
  - podem conter campos, métodos e tipos como membros.

Aspectos versus classes

- Aspectos são diferentes de classes:
  - não têm construtor, nem destrutor;
  - não podem ser criados com o operador `new`;
  - podem conter `pointcuts` e `advices` como membros;
  - podem acessar membros de outros tipos.

Joinpoints

- São pontos na execução de um programa, onde aspectos podem interceptar classes.



Joinpoints

- Joinpoints são identificados por designadores:
  - `call(`Signature`)`
  - `execution(`Signature`)`
  - `initialization(`Signature`)`
  - `handler(`TypePattern`)`
  - `get(`Signature`)`
  - `set(`Signature`)`
  - `this(`TypePattern`)`
  - `target(`TypePattern`)`
  - `args(`TypePattern, ...`)`
  - `within(`TypePattern`)`
  - `cflow(`PointCut`)`

Joinpoints

- Designadores podem ser genéricos:
  - `call(* set*())`
  - `call(public Person.*(..))`
  - `call(void foo(..))`
  - `call(* *(..))`
  - `call(*.new(int, int))`

Pointcuts

- Predicados que definem conjuntos de joinpoints.

```
pointcut traced() :
  call(public * set*(..));
```

Pointcuts

- Pointcuts podem ser definidos como expressões booleanas, empregando-se !, && e ||.

```
pointcut move():
   call(Point.setX(..)) ||
   call(Point.setY(..)) ||
   call(Line.setP1(..)) ||
   call(Line.setP2(..));
```

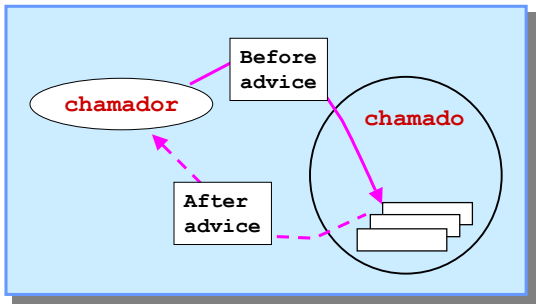

Pointcuts

- Pointcuts também podem ter argumentos.

```
pointcut move(Object m, Shape s):
   this(m) && target(s) &&
   (call(Point.setX(..)) ||
    call(Point.setY(..)) ||
    call(Line.setP1(..)) ||
    call(Line.setP2(..)));
```

Advices

- São trechos de código associados a pointcuts, que injetam um novo comportamento em todos os joinpoints representados pelo pointcut.
- Tipos:
  - `before`
  - `after`
  - `around`



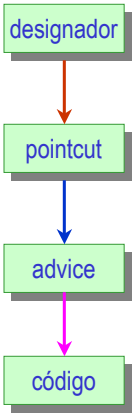
Advices

- **Formas básicas:**
  - `before(param) : pointcut(param) {...}`
  - `after(param) : pointcut(param) {...}`
  - `after(param) returning [formal] : pointcut(param) {...}`
  - `after(param) throwing [formal] : pointcut(param) {...}`
  - `type around(param) [throws typelist] : pointcut(param) {...}`

Fluxo de informação

```
pointcut get(int index) :
  args(index) && call(*get*(..));

before(int index) : get(index) {
     System.out.println(index);
}
```



Exemplo: Hello world!

```
public class Hello {
   public static void main(String args[]) {
     System.out.println("Hello world!");
   }
}
```

```
public aspect Greetings {
   pointcut pc() : call(* *main(..));
   before() : pc() { System.out.println("Hi."); }
   after() : pc() { System.out.println("Bye..."); }
}
```

Exemplo: Hello world!

```
>ajc Hello.java Greetings.java
>java Hello

Hi.
Hello world!
Bye...
```


Exemplo: Hello world!

```
/*   Generated by AspectJ version 1.0.6 */
import java.io.*;
public class Hello {
  public static void main(String[] args) {
    try {
       Greetings.aspectInstance.before0$ajc();
       Hello.main$ajcPostCall(args);
    } finally {
       Greetings.aspectInstance.after0$ajc();
    }
  }
  public Hello() {
    super();
  }
  public static void main$ajcPostCall(String[] args) {
    System.out.println("Hello world!");
  }
}
```

Exemplo: Hello world!

```
/*   Generated by AspectJ version 1.0.6 */
public class Greetings {
  public final void before0$ajc() {
    System.out.println("Hi.");
  }
  public final void after0$ajc() {
    System.out.println("Bye...");
  }
  public Greetings() {
    super();
  }
  public static Greetings aspectInstance;
  public static Greetings aspectOf() {
    return Greetings.aspectInstance;
  }
  public static boolean hasAspect() {
    return Greetings.aspectInstance != null;
  }
  static {
    Greetings.aspectInstance = new Greetings();
  }
}
```


Reflexão: `thisJoinPoint`

- É uma variável especial predefinida.
- É similar à variável `this` em Java.
- Disponibiliza várias informações a respeito do joinpoint que disparou o advice.
- Exemplos:
  - `thisJoinPoint.toString()`
  - `thisJoinPoint.getArgs()`

Introduction

- Um forma de introduzir novos membros a classes e interfaces já existentes.

```
public int Foo.bar(int x);
private int Foo.counter;
declare parents: Mammal extends Animal;
declare parents: MyThread implements
MyThreadInterface;
```


Reusabilidade

```
public abstract aspect Tracing {
  protected abstract pointcut trace();
  // advice code
}

public aspect MyTracing extends Tracing {
   // just define which calls to trace
   protected pointcut trace(): call(* set*());
}
```

IME - USP

Exemplos

Rastreamento
Sincronização
Otimização

Rastreamento

```
aspect Tracing {
  pointcut traced() : call(public * Person.*(..));
  before() : traced() {
    System.err.println("entry: " + thisJoinPoint);
  }
  after() : traced() {
    System.err.println("exit: " + thisJoinPoint);
  }
}
```

```
class Person {
  private String name = "";
  public void setName(String name) {
    this.name = name;
  }
  ...
}
```

Sincronização

```
aspect Synchronization {
  private Semaphore s = new Semaphore();
  pointcut updateSem(): call(public void set*(..));
  before() : updateSem() { P(s); }
  after()  : updateSem() { V(s); }
}
```


Otimização

```
public class Fibonacci {
  public static int fib(int n) {
    if( n < 2 ) {
      System.err.println(n + ".");
      return 1;
    }
    else {
      System.err.print(n + ",");
      return fib(n-1) + fib(n-2);
    }
  }
  public static void main(String[] args) {
      int f = fib(10);
      System.err.println("Fib(10) = " + f);
  }
}
```

Otimização

```
aspect Memoization {

  private HashTable cache = new HashTable();

  pointcut fibs(int n):
    execution(int Fibonacci.fib(int)) && args(n);

  // calculate only if not already in cache!

  int around(int n): fibs(n) {
    Integer result = (Integer)cache.get(new Integer(n));
    if (result == null) {
      int f = proceed(n); // not found, calculate!
      cache.put(new Integer(n), new Integer(f));
      return f;
    }
    else return result.intValue(); // found, done!
  }
}
```


Referências

- Kendall, E. A. *Aspect-Oriented Programming in AspectJ*, 2001.
- Kiczales, G. et all. *Aspect-Oriented Programming*, 1997.
- Kiczales, G. et all. *An Overview of AspectJ*, 2002.
- Voelter, M. *Aspect-Oriented Programming in Java*, 2000.
- *The AspectJ Programming Guide*, Xerox Corporation, 2002.

IME - USP

Fim