

SUMÁRIO - AULA 09

- ❑ Introdução à alguns paradigmas

- ❑ *Aspect-Oriented Programming (AOP)*

- ❑ Programação orientada a Aspectos (POA)



PROBLEMA

- ❑ Você implementou **todas** as funções de um **grande** sistema de controle de acesso com **êxito...oba!**

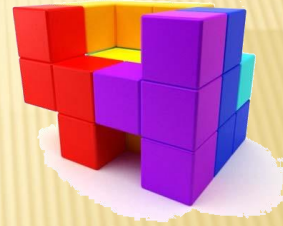
- ❑ Agora o cliente quer registrar **todas** as **alterações** que ocorrem nos **cadastros...**

- ❑ O que fazer?

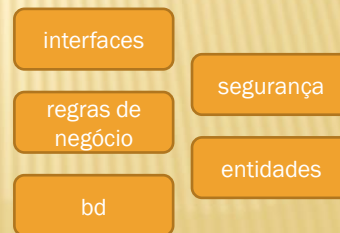


INTRODUÇÃO A POA

- Um **sistema** comumente é **dividido** em **partes** menores tratadas separadamente.



- reduzir** a complexidade
- separação de **interesses**
- modularidade** do sistema



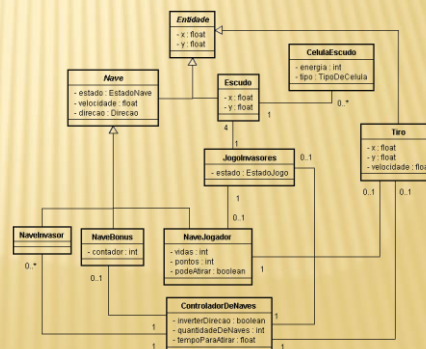
INTRODUÇÃO A POA

- Na POO o modelo de **abstração** trabalha com **classes**.

- elas modelam os **dados + funções**

- No **projeto** procura-se **separar** cada **interesse** em uma **classe** distinta.

- mas nem sempre isso é possível...



INTRODUÇÃO A POA

- ❑ Para aqueles **interesses** que **não** podem ser **modularizados** em classes dá-se o nome de:

- ❑ **interesses transversais**
- ❑ *interesses entrecortantes*
- ❑ *crosscutting concern*



O log, por exemplo, é uma funcionalidade que aparece em inúmeros locais de um sistema.

- ❑ Estão **dispersos/pulverizados** no sistema.

INTRODUÇÃO A POA

- ❑ Exemplos típicos:

- ❑ persistência de objetos
- ❑ segurança de acesso
- ❑ acesso concorrente
- ❑ gerenciamento de transações
- ❑ geração de log
- ❑ auditoria
- ❑ tratamento de exceções
- ❑ inicialização de objetos
- ❑ e muitos outros...



Não existem tais módulos já que estas funções estão **dispersas em todo o sistema**



INTRODUÇÃO A POA

□ “Intrusos” no código não são bem vindos...

□ Entidades que não resolvem o problema, que destoam dos demais envolvidos.

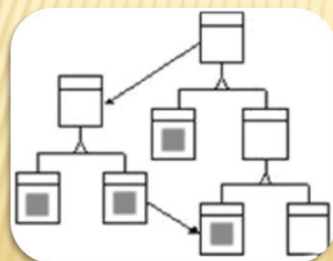
INTRODUÇÃO A POA

- ❑ Faremos um “sobrevoo” no código

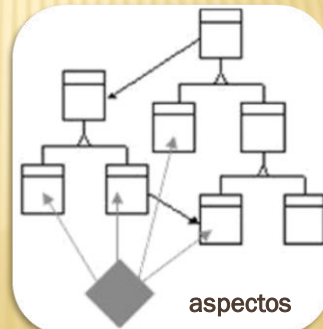


INTRODUÇÃO A POA

- ❑ O objetivo da POA é **encapsular interesses entrecortantes** em módulos **fisicamente separados** do restante do código.
 - ❑ Esses módulos são denominados **aspectos**.



interesses transversais



aspectos



INTRODUÇÃO A POA



- Ao desenvolver com POA, a **lógica de controle** é **invertida**

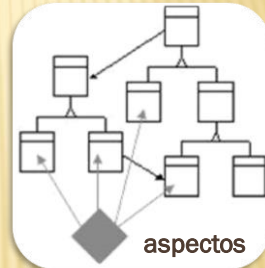
- *Inversion of Control – IoC*

- Os aspectos **conhecem** os componentes do sistema...

- Mas os componentes **desconhecem** os aspectos.

- princípio de hollywood:

- "Don't call us, we'll call you"



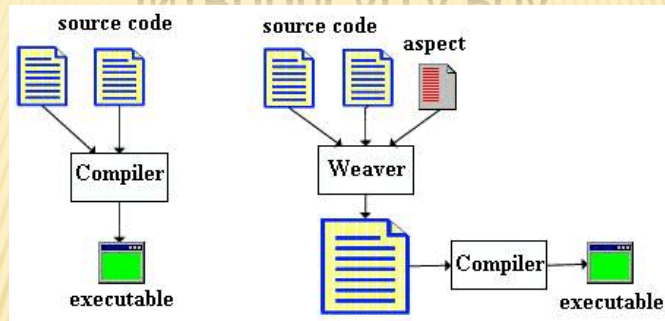
INTRODUÇÃO A POA



- Para **compor** o sistema é preciso fazer o processo de **montagem**, que consiste em aplicar os **aspectos sobre** os componentes.

- Este processo é chamado **combinação** (weaving) sendo realizado por um **combinador** (weaver).

INTRODUÇÃO A POA



- ❑ O weaver pode ser
 - ❑ **estático**
 - ❑ aplicado em tempo de **build** do sistema
 - ❑ **dinâmico**
 - ❑ aplicado durante a **execução** do código.

INTRODUÇÃO A POA

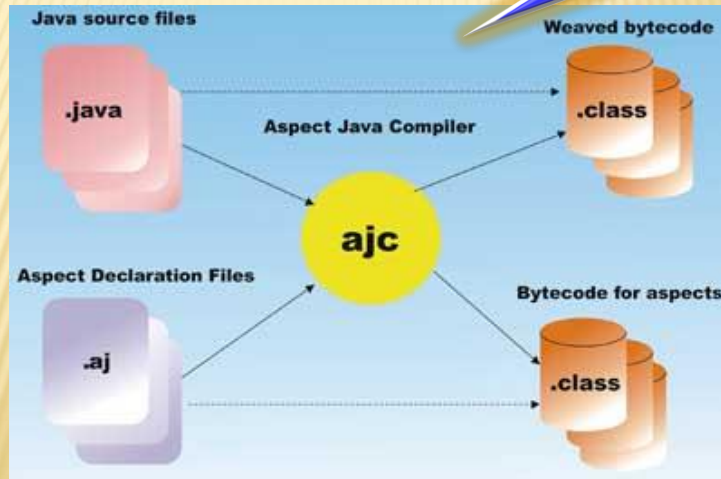


- ❑ O **AspectJ** é um conjunto de **compilador**, **combinador** e utilitários de POA para Java.
 - ❑ <http://eclipse.org/aspectj>
- ❑ A **sintaxe** utilizada no AspectJ para a construção dos aspectos é **semelhante** a **Java** (extensão)
 - ❑ um dos motivos de sua ampla aceitação.

INTRODUÇÃO A POA

□ AJC: Aspect Java Compiler

ajc compila .java e .aj, gerando .class

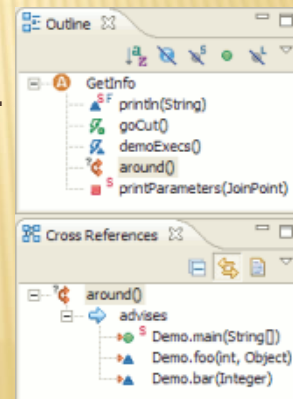


INTRODUÇÃO A POA

□ A **ferramenta** que será usada para demonstrar as técnicas de POA é o **AJDT** do projeto Eclipse.

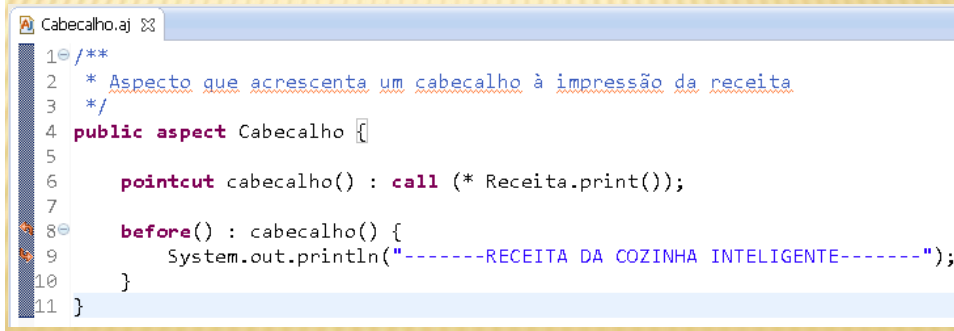
□ **Plug-in** integrado ao IDE Eclipse.

□ <http://eclipse.org/ajdt>



PRINCIPAIS CONCEITOS NA POA

- ❑ **aspectos** são semelhantes a **classes**
 - ❑ definidos em arquivos com extensão **.aj**
 - ❑ sintaxe **semelhante** da linguagem Java

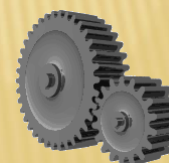


```

1 /**
2  * Aspecto que acrescenta um cabeçalho à impressão da receita
3  */
4  public aspect Cabecalho {
5
6      pointcut cabecalho() : call (* Receita.print());
7
8      before() : cabecalho() {
9          System.out.println("-----RECEITA DA COZINHA INTELIGENTE-----");
10     }
11 }
  
```

PRINCIPAIS CONCEITOS NA POA

- ❑ **joinpoints (j)**
 - ❑ trecho **alvo** do código fonte
 - ❑ “método XYZ”
 - ❑ “atributos VWT”
- ❑ **pointcuts (p)**
 - ❑ meio de **definir/identificar** os joinpoints
 - ❑ agrupados via regras
 - ❑ “monitorar todos os **métodos** iniciados com X”
- ❑ **advices (a)**
 - ❑ **procedimentos** executados quando os pointcuts são disparados nos joinpoints
 - ❑ “quando X for executado, faça isso...”





PRINCIPAIS CONCEITOS NA POA

- ❑ **joinpoints** - **locais/eventos** de interesse do fluxo de execução.
- ❑ Quando a execução passa por um **joinpoint** o aspecto pode **agir** naquele ponto.
- ❑ Exemplo de **joinpoints**:
 - ❑ invocação de métodos
 - ❑ alteração de atributos
 - ❑ auditoria
 - ❑ disparo de exceções.



PRINCIPAIS CONCEITOS NA POA

- ❑ **pointcuts** - usados para **representar** um **conjunto** de joinpoint
- ❑ podem acontecer **muitas ocorrências** de joinpoints de um mesmo tipo (ex: get) e então **expressões regulares** podem ser usadas na **definição** de pointcuts.

```
public String getNomeReceita() {
    return nomeReceita;
}

public void setNomeReceita(String nomeReceita) {
    this.nomeReceita = nomeReceita;
}
```



PRINCIPAIS CONCEITOS NA POA

- **pointcuts** – simular o uso de um comando SQL para retornar todos os métodos que iniciam com **get**.
- *SELECT * FROM código WHERE nome_do_método LIKE “get%”*



PRINCIPAIS CONCEITOS NA POA

- definição de **pointcuts**

```
public String getNomeReceita() {
    return nomeReceita;
}

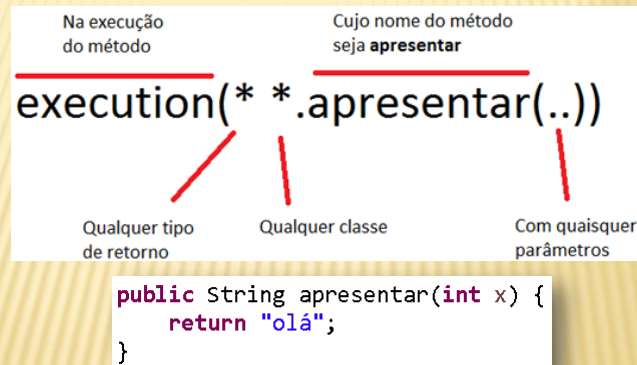
public void setNomeReceita(String nomeReceita) {
    this.nomeReceita = nomeReceita;
}
```

Any return type package class method any type and number of arguments

```
@Pointcut("execution(* aspects.trace.demo.*(..))")
public void traceMethodsInDemoPackage() {}
```

PRINCIPAIS CONCEITOS NA POA

- **pointcuts** – sintaxe um tanto rebuscada...



- Mais informações em:

- <http://www.eclipse.org/aspectj/doc/next/progguide/se-mantics-pointcuts.html>



PRINCIPAIS CONCEITOS NA POA

- **advices** - são os **procedimentos/ações** realizados quando os pointcuts são **ativados**.
 - Os **advices** podem ser executados antes (*before*), depois (*after*) ou em substituição (*around*) ao **joinpoint**.
 - **advices** são a **implementação** dos interesses transversais.
 - O serviço a ser feito.

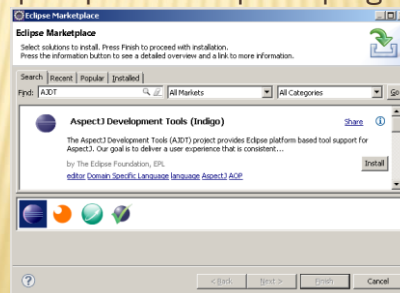
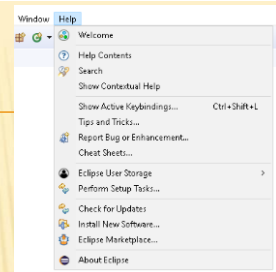
RESUMO DOS CONCEITOS



- ❑ **ponto de junção (*join points*)**
 - ❑ qualquer ponto de execução identificável num programa. Ex:
 - ❑ chamada a métodos, construtores, variáveis, etc.
- ❑ **ponto de corte ou secção (*point cut*)**
 - ❑ identificam e capturam pontos de junção de um programa. Regras como “*queries*”.
- ❑ **conselhos (*advices*)**.
 - ❑ especificar as operações/ações

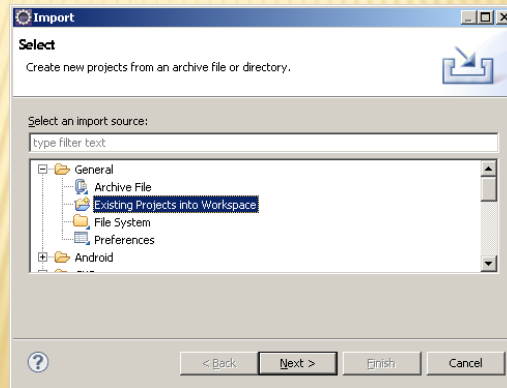
POA NA PRÁTICA

- ❑ 1) Instalar AJDT no Eclipse.
 - ❑ via update site:
 - ❑ <http://download.eclipse.org/tools/ajdt/47/dev/update>
 - ❑ via market place
 - ❑ menu Help | Eclipse Marketplace | digitar AJDT e buscar



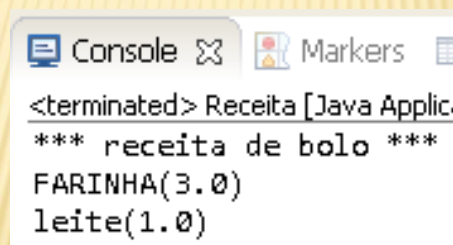
POA NA PRÁTICA

- ❑ 2) Importar via **SIGAA** projeto (.zip) da Aula 09
 - ❑ acessar menu *File | Import | General | Existing Projects into Workspace*



ATIVIDADE

- ❑ 3) Execute o projeto via a classe **Executar**, inicialmente **sem** suporte da POA.
 - ❑ O que faz tal projeto?

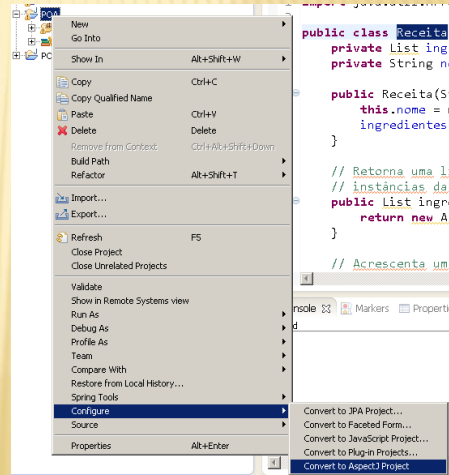


- ❑ Exemplo alterado de:
 - ❑ <http://www.aspectos.org/courses/aulasaop/aula07.html>

ATIVIDADE

❑ 4) Adicione suporte a POA ao projeto:

- ❑ Para tal, selecione o projeto e converta-o em *AspectJ* project



ATIVIDADE

❑ 5) Execute a classe Executar.

- ❑ O que aconteceu?
- ❑ O que fazem os aspectos?
- ❑ Onde atuam?



ATIVIDADE



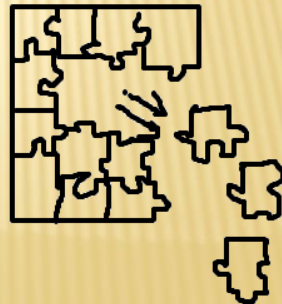
- ❑ 6) Agora **altere** o projeto e **explore** as funcionalidades da POA. Sugestões:
 - ❑ adicione **novas classes, métodos** e sobre eles defina *pointcuts*, *joinpoints*, *advices*, etc.
 - ❑ mostre o **horário** em que cada método foi executado, valores **antes** e **depois** do processamento pelo método, etc.
 - ❑ crie **receitas**, capture exceções, gere log, etc.

ATIVIDADE

- ❑ 7) **Descompile** algum arquivo **.class** gerado pelo **AJC** e veja se encontra as instruções relativas aos aspectos que foram incorporadas.

- ❑ Sugestão de :

❑ <http://jd.benow.ca>




```

1 public class Executar {
2
3
4 public static void main(String[] args) {
5     Receita r = new Receita("bolo");
6     r.adicionaIngrediente(new Ingrediente("FARINHA", 3));
7     r.adicionaIngrediente(new Ingrediente("leite", 1));
8     r.print();
9 }
10 }

```

Executar.java

Classes compiladas

- ☒ Alt.class
- ☒ Cabecalho.class
- ☒ Executar.class
- ☒ Ingrediente.class
- ☒ NormalizaCaixa.class
- ☒ Receita.class

```

public class Executar
{
    public Executar()
    {
    }

    public static void main(String args[])
    {
        Receita r = new Receita("bolo");
        r.adicionaIngrediente(new Ingrediente("FARINHA", 3F));
        r.adicionaIngrediente(new Ingrediente("leite", 1.0F));
        Cabecalho.aspectOf().ajc$before$Cabecalho$1$30004d5();
        r.print();
    }
}

```

Executar.class

```

import java.io.PrintStream;
import org.aspectj.lang.NoAspectBoundException;

@Aspect
public class Cabecalho
{
    public Cabecalho()
    {
    }

    @Pointcut(value="call(* Receita.print())", argNames="")
    void ajc$pointcut$Cabecalho$71()
    {
    }

    @Before(value="cabecalho()", argNames="")
    public void ajc$before$Cabecalho$1$30004d5()
    {
        System.out.println("-----RECEITA DA COZINHA INTELIGENTE-----");
    }

    public static Cabecalho aspectOf()
    {
        if(ajc$perSingletonInstance == null)
            throw new NoAspectBoundException("Cabecalho", ajc$initFailureCause);
        else
            return ajc$perSingletonInstance;
    }
}

```

Cabecalho.class

sim, aspectos se tornam classes compiladas (.class)
e os binários de nossas classes são alterados!

POA NA PRÁTICA

❑ Para saber mais:



- ❑ http://www.cin.ufpe.br/~acnlf/curso_poa.pdf
- ❑ <https://o7planning.org/en/10257/java-aspect-oriented-programming-tutorial-with-aspectj>

ATIVIDADE

- ❑ Existem **ferramentas** para aplicar POA em outros ambientes e linguagens de programação?
- ❑ Pesquise a respeito.



ATIVIDADE

- ❑ Conhecem *Reflection* e *Annotations*?
- ❑ <http://www.caelum.com.br/apostila-java-testes-xml-design-patterns/reflection-e-annotations>

