# JBoss AOP

## Interface para o Usuário

# Principais conceitos

- Aspectos
- Advices
- Pointcuts
- Introductions de interfaces e Mixins
- Joinpoints
- Invocations

# Definindo Aspectos

- No JBoss AOP, um aspecto é uma classe;
- Você precisa declarar os seus aspectos em declarações em um arquivo xml:

```
<aspect class="AspectPerVM" scope="PER_VM"/>
```

# Utilizando advices

- Um *advice* é um método de um aspecto;
- Deve ser da forma:

```
public Object <qualquer-
   nome>(<invocation>) throws Throwable
```

- Não é preciso declarar o seu *advice*; apenas utilize-o na interceptação de *pointcuts*.

# Interceptadores

- Um interceptador é um aspecto com um único *advice*;
- A assinatura do interceptador é definida pela interface

  **`org.jboss.aop.advice.Interceptor`:**

```
public Object invoke(Invocation i) throws
Exception
```

# Interceptações

- Para interceptar um *pointcut* é preciso declarar um *binding*.
- Em um *binding*, você diz qual a expressão *pointcut*, e qual a pilha de interceptadores:

```
<bind pointcut=
"execution(public void POJO->noop())">
<interceptor class="SimpleInterceptor"/>
<advice name="methodAdvice" aspect="MyAspect"
  />
</bind>
```

# Configurações com Anotações

```
@Aspect
public class MyAspect {
  @Bind (pointcut="execution(POJO->new())")
  public Object constructorAdvice(
       ConstructorInvocation invocation)
      throws Throwable
  { ... }

  @Bind (pointcut="execution(void POJO->method())")
  public Object methodAdvice(
    MethodInvocation invocation)
    throws Throwable { ... }
}
```

# Outros recursos

- Introdução de Interfaces e Mixins
- Introdução de Anotações
- Pointcuts Plugáveis
- Definições de Tipo
- CFlow
- CFlow Dinâmico
- Pilhas de CFlow e de Advices
- Precedência
- Preparação
- Anotações Sobrescritas
- Metadados
- Declaração de Aviso ou de Erro

# Como executar o JBoss AOP

- Utilizando a tarefa ant aopc para processar as suas classes compiladas:
  - Alternativamente, podemos executar:

```
java org.jboss.aop.standalone.Compiler
```

- Utilizando agentes do Java 5, que será detalhado na próxima apresentação, dia 13/06/2005

# JBoss AOP

Uma visão da sua implementação

# Como o JBoss AOP Funciona?

- Objetivo: habilitar a programação orientada a aspectos ou POA;
  - O usuário define os pontos do código a serem interceptados;
  - O usuário espera que, durante a execução do seu sistema, tais pontos sejam interceptados.

# Como o JBoss AOP Funciona?

- É preciso que:
  - Exista uma forma de o usuário dizer ao JBoss AOP o que ele deseja;
  - As classes do usuário sejam alteradas de alguma forma para que a interceptação seja codificada;
  - Durante a execução, o código interceptado entra em ação, e as classes são interceptadas do modo que o usuário definiu.

# Três Etapas

- Configuração
- Instrumentação ou Transformação
- Interceptação

# A classe AspectManager

- `org.jboss.aop.AspectManager` é a classe principal do sistema.
  - Fachada para o JBoss AOP;
  - Guarda todos as informações configuradas.
- Para executar a primeira etapa, configuração, basta que o método seja invocado:

  `org.jboss.aop.AspectManager.instance()`

# A classe AspectManager

- Isso significa que não importa a forma como o JBoss AOP seja executado (através de AOP ou através de um agente de Java 5), a primeira etapa (configuração) será realizada simplesmente obtendo-se a instância *singleton* de **`AspectManager`**.

# A classe AspectManager

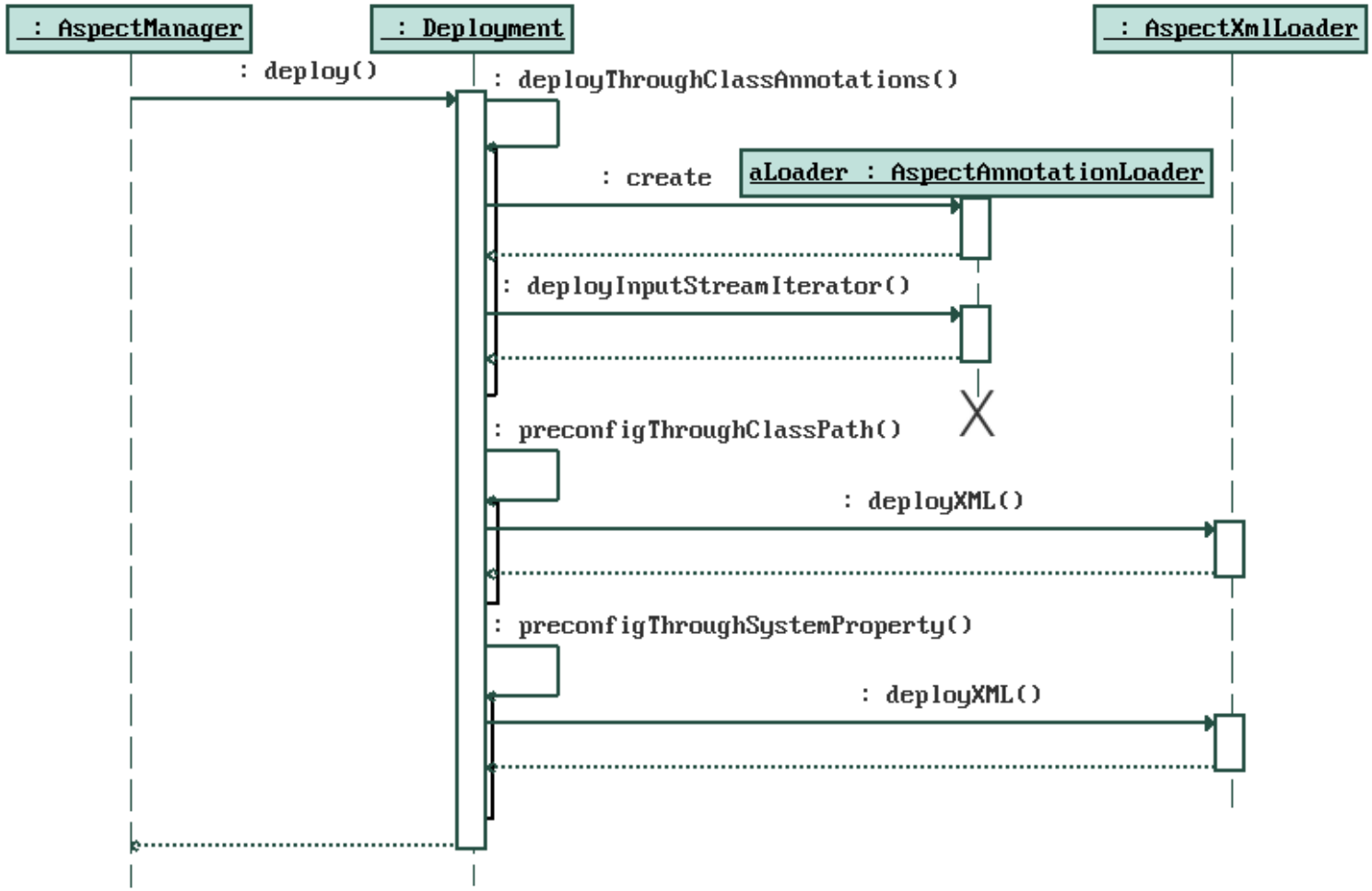
- Para realizar a segunda etapa, transformação, **`AspectManager`** fornece o seguinte método, que deve ser invocado para cada classe a ser transformada:

```
transform(ClassLoader cl, String
  className, Class classBeingRedefined,
  ProtectionDomain domain, byte[]
  classFileTransformer)
```

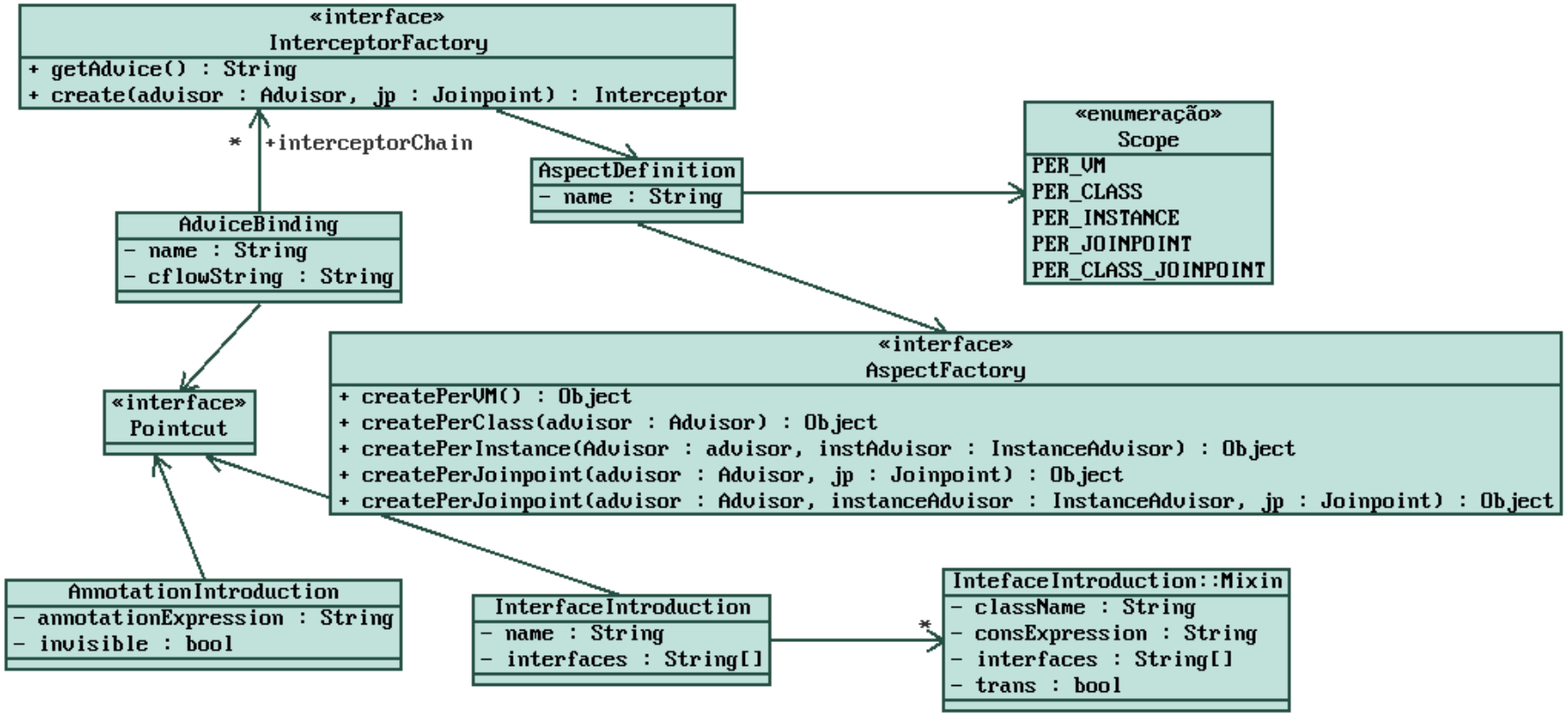
# Parte I: A Configuração

- A configuração é realizada através de duas classes:
  - **`AspectAnnotationLoader`**
    - Carrega as informações contidas em anotações.
  - **`AspectXmlLoader`**
    - Carrega as configurações feitas através de arquivos xml (tipicamente denominados jboss-aop.xml).

# Parte I: A Configuração



# A Estrutura de Classes



# Anotações

- JBoss AOP lida com anotações de uma forma interessante.
- Existem duas versões do JBoss: uma para Java 4, e outra para o Java 5;
- Apesar disso, existe um esforço para que o número de classes comuns entre as duas versões seja o maior possível;

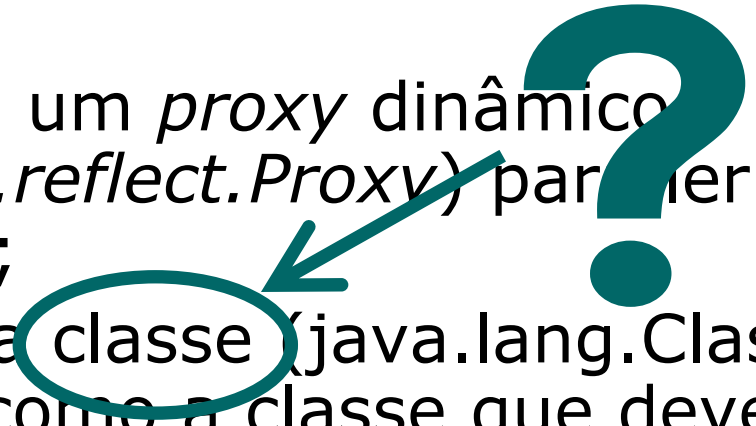
# Anotações

- Com o JBoss AOP um usuário que utiliza Java 4 pode se beneficiar do uso de anotações, através de *doclets*.
- O algoritmo que processa as classes para ler as anotações é o mesmo para as duas versões.

# Anotações em Java 4

- Utilizar *doclets* para anotar as classes;
- Processar as classes com a ferramenta annotationc.
- Arquivos .class com anotações nos bytecodes são gerados.

# Leitura de Anotações



- Criação de um *proxy* dinâmico (*java.lang.reflect.Proxy*) para ler as anotações;
- Ele passa a classe (java.lang.Class) da anotação como a classe que deve ser implementada;
- E o proxy handler (org.jboss.aop.annotation.AnnotationProxy) é responsável por conhecer os atributos da anotação e responder pelos métodos.

# As classes das anotações

- No Java 5:

```
@Target({ElementType.TYPE})
@Retention(RetentionPolicy.RUNTIME)
public @interface Aspect{
    Scope scope() default Scope.PER_VM;
}
```

- Java 4:

```
public interface Aspect {
   Scope scope();
}
```

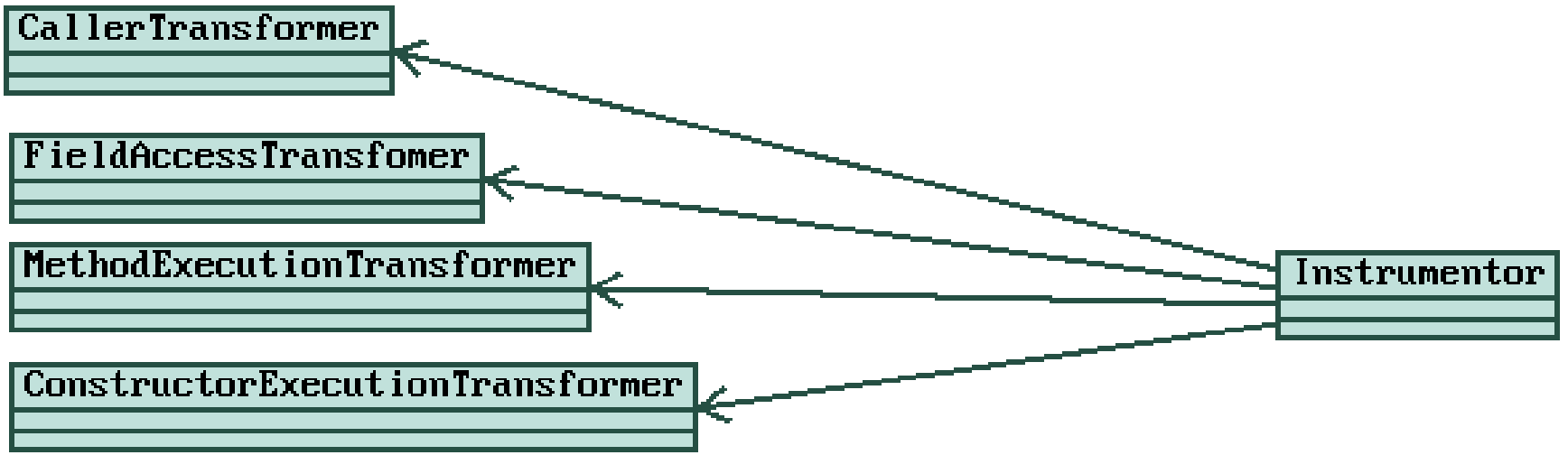
# Parte II: A Instrumentação

- É a etapa que segue a configuração.
- É realizada pelo método: AspectManager.transform
- É preciso verificar se a classe não é de alguma biblioteca que não deve ser alterada, como java.lang.String, org.jboss.aop.Interceptor e etc.

# Parte II: A Instrumentação

- A instrumentação é delegada para org.jboss.aop.instrument.Instrumentor, fachada para o pacote org.jboss.aop.instrument.

# O pacote org.jboss.aop.instrument



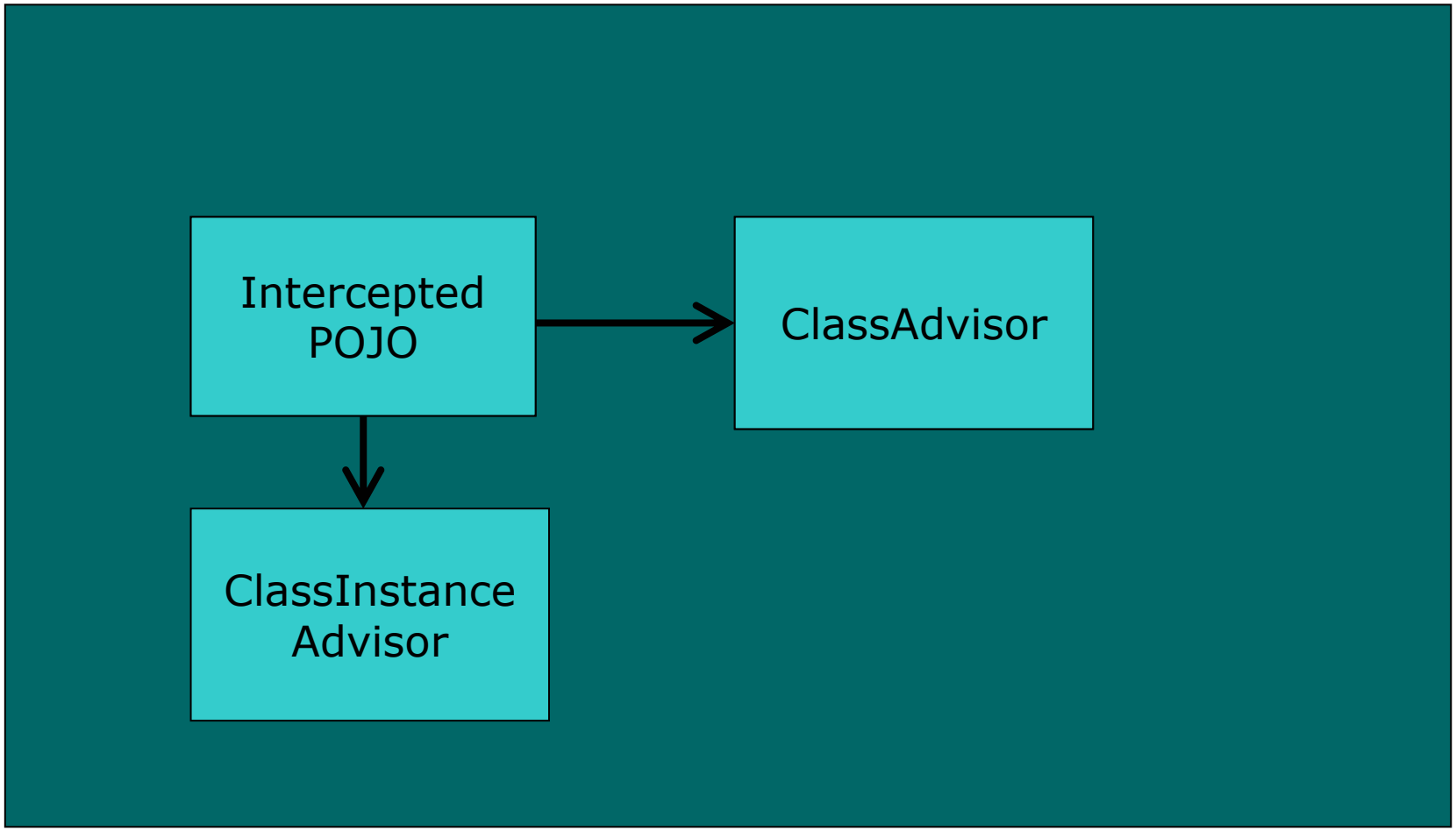
# Mas o que é a instrumentação?

- Primeiro passo, a adição de meta-dados (veremos mais tarde);
- Busca das introduções de interfaces que deverão ser aplicadas;
- Verificação das declarações de erro e de aviso;

# Mas o que é a instrumentação?

- E, finalmente, a alteração de *bytecodes*:
  - Cada transformador verifica se a classe possui algum *joinpoint* que deverá ser interceptado;
  - Se um *joinpoint* for encontrado, a classe deve ser associada a um Advisor.
  - Após isso o *joinpoint* é instrumentado para a interceptacao.

# Advisors



# Associação a um Advisor

- A classe passa a implementar a interface Advised:

```
private static final ClassAdvisor
  aop$classAdvisor$aop =
  AspectManager.instance().getAdvisor
  (Class.forName("myPackage.MyClass")
  );
public Advisor _getAdvisor()
{ return aop$classAdvisor$aop; }
```

# Associação a um Advisor

- Implementa a interface InstanceAdvised:

```
protected transient ClassInstanceAdvisor
   _instanceAdvisor;
public InstanceAdvisor_getInstanceAdvisor()
{
   synchronized (this) {
       if (_instanceAdvisor == null) {
             _instanceAdvisor = new
             ClassInstanceAdvisor(this);
       }
   }
}
```

# Instrumentação para Interceptação

- Os *joinpoints* são encapsulados (*wrapped*) em um código que executa a pilha de interceptadores:

```
wrapper() {
 para cada interceptador
    executa o interceptador
 executa o joinpoint
}
```

# Instrumentação de Construtores

```
public class MyClass {
 public MyClass()
 {
   System.out.println("Hi");
 }
public static MyClass
       MyClass$aop$()
{ // apply interceptors
 return new MyClass();}
}
```

```
public class OtherClass {
public someMethod()
{
...
MyClass mc =
MyClass.MyClass$aop$();
...
}
```

# Instrumentação de Campos

```
public class MyClass {
 public int myField;

public static void
  myField_w_$aop(Object obj,
   int value){
    // apply interceptors
    obj.myField = value;
 }
}
```

```
public class OtherClass {
  public someMethod()
  {
   MyClass mc = ...;
   MyClass.myField_w$aop
         (mc, 10);
   ...
}}
```

# Instrumentação de Métodos

```
public class MyClass {
  public void myMethod() {
    // apply interceptors
    this.MyClass$myMethod$aop();
   }

  public void
   MyClass$myMethod$aop() {
   System.out.println("Hi");
  }
}
```

```
public class OtherClass {
public someMethod()
{
 MyClass mc = ...;
 mc.myMethod();
 ...
}
```

# Instrumentação de Chamadas

```
public class MyClass {
 public void myMethod() {
  System.out.println("Hi");
 }
}
```

```
public class OtherClass1 {
  public someMethod(){
    MyClass mc = ...;
    {
    // apply interceptors
    mc.myMethod();
    }
    ...
  }
}
```

```
public class OtherClass2 {
  public someMethod(){
    MyClass mc = ...;
    mc.myMethod();
   ...
   }
}
```

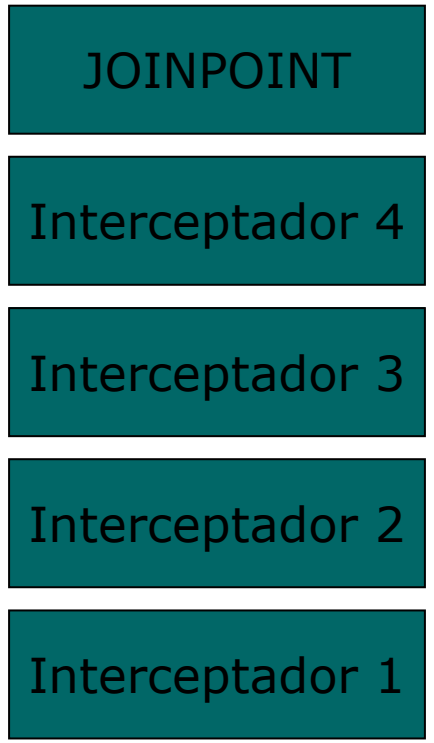
# A classe Invocation

- Você se lembra disso:

```
wrapper() {
    para cada interceptador
        executa o interceptador
    executa o joinpoint
}
```

- Na verdade a coisa é um pouco diferente:

```
Cria o Invocation apropriado
Retorna invocation.invokeNext()
```



# O Invocation a ser Utilizado

- O JBoss AOP possui várias implementações de Invocation:
  - FieldRead, MethodExecution, ConstructorCalledByConstructor, etc;
  - Todas elas executam a pilha de interceptadores a cada invokeNext();
  - E depois disso, executam o joinpoint.
    - Isso difere a cada classe, e é feito com *reflection*.

# Invocations Otimizadas

- São classes geradas durante a transformação;
- Elas simplesmente invocam o *joinpoint*, sem o uso de reflexão, pois têm que ser genéricas;

# Introdução de Interfaces

```
public class MyInterface{
  public void myMethod();
}
```

```
public class MyClass
  implements MyInterface {
 public someMethod() {..}
private MyMixin mixin =
    new MyMixin();

 public void myMethod() {
   mixin.myMethod();
 }
}
```

```
public class MyMixin
  implements MyInterface{

public void myMethod() {
  System.out.println("Hi");
 }
}
```

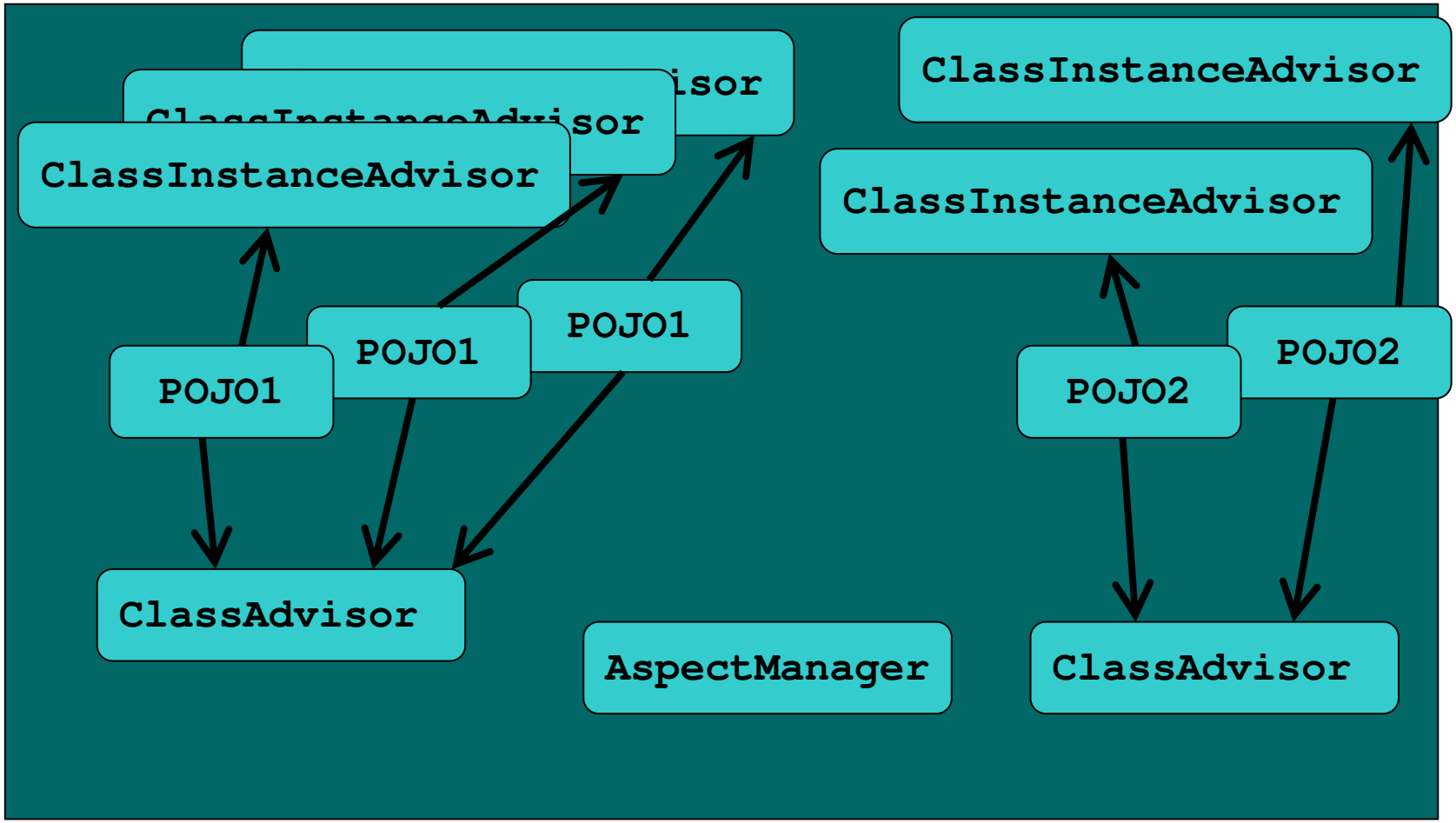
# Outras transformações

- Introduções de anotações
- Declarações de erros e de avisos

# Parte III: A interceptação

- Ocorre quando as classes instrumentadas forem executadas;
- Já vimos boa parte do código que será executado durante essa fase;
- Nessa fase, instâncias dos aspectos e interceptadores serão criadas para serem invocadas durante a interceptação de um joinpoint.

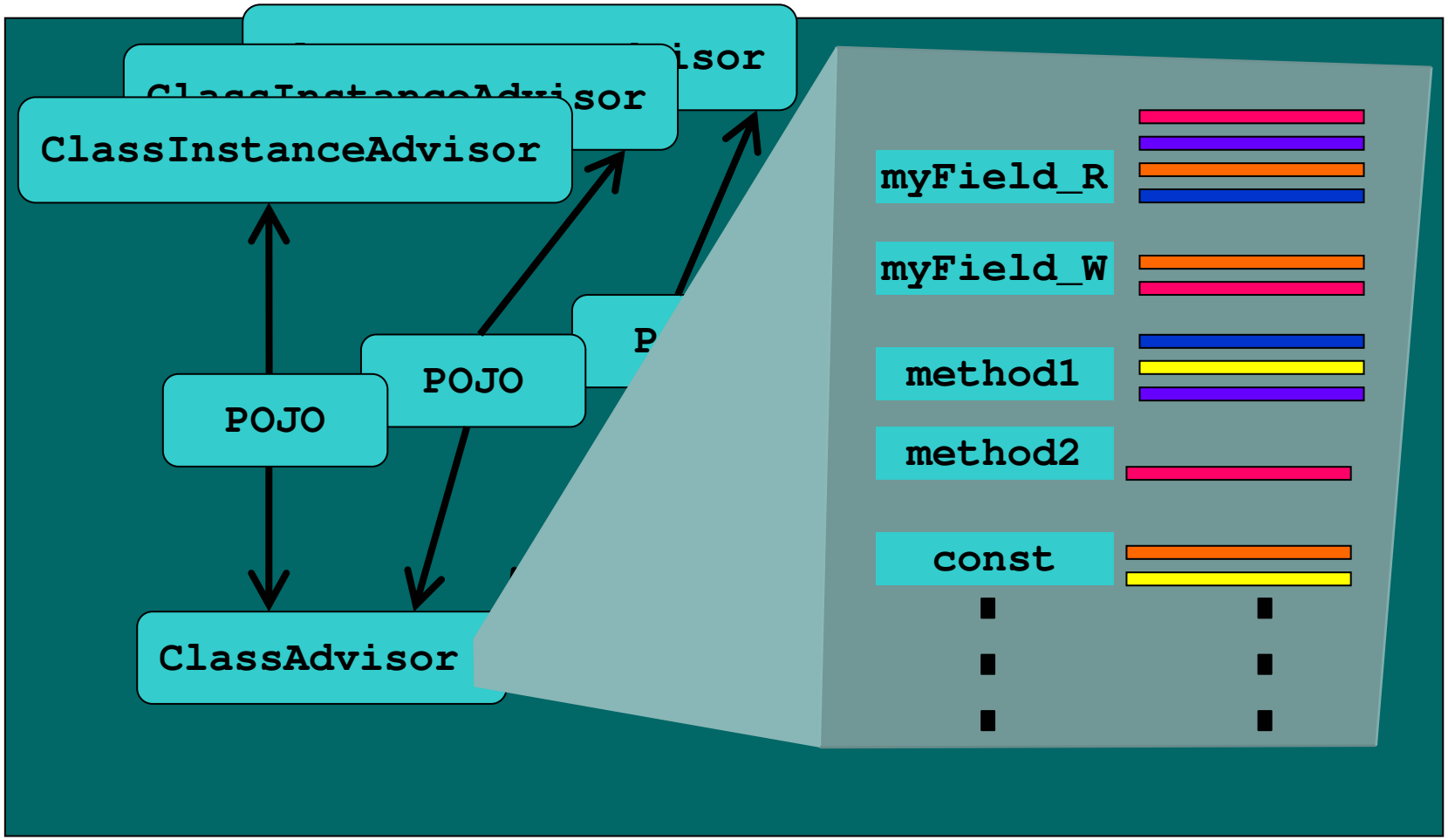
# Fábricas de Aspectos



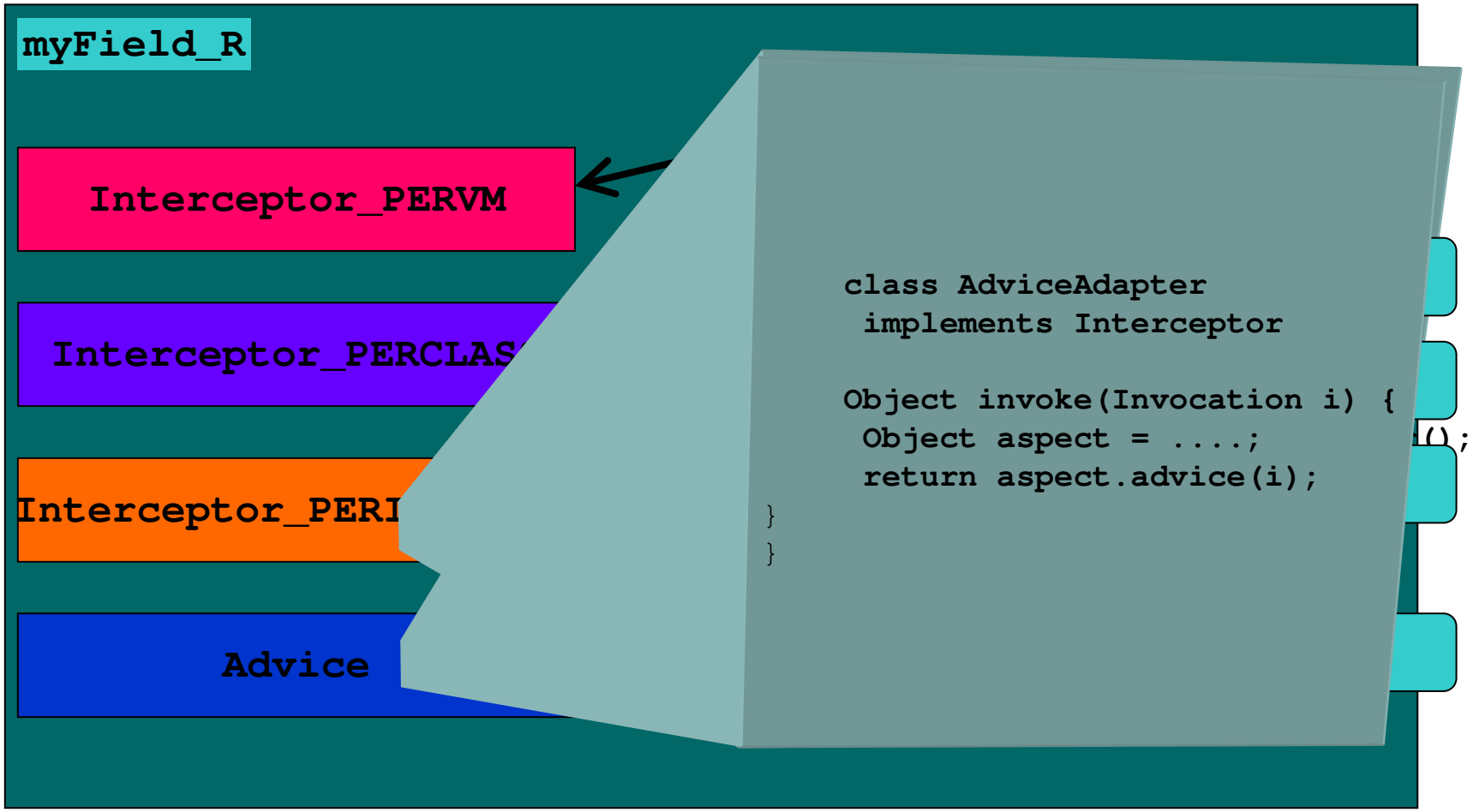
# ClassAdvisor

- Lembra dessa linha:
  - AspectManager.instance().getAdvisor(Class.forName(myPackage.MyClass))
- O que realmente acontece:
  - AspectManager cria um ClassAdvisor;
  - O advisor:
    - aplica os meta-dados;
    - encontra os joinpoints da classe;
    - Monta uma pilha de interceptadores para cada joinpoint.

# ClassAdvisor



# Pilhas de Interceptadores



# Obtenção de Anotações

- O cliente pode obter anotações das formas:
  - Através de reflexão no java 5;
  - Através do Advisor (métodos resolveAnnotation e hasAnnotation)
    - Compatível com Java 4;
    - É a única forma de acessar anotações sobrescritas;
    - Retorna o proxy dinâmico.

# Meta-Dados

- Permite a associação de meta-dados (informações sem tipagem definida) a uma classe;
- São acessados através do Advisor e do Invocation;
- Serve para:
  - Comunicação entre *advices*;
  - Configuração em arquivos xml para advices;

# Meta-Dados

```
<metadata tag="testdata"
   class="org.jboss.test.POJO">
   <default>
        <some-data>default value</some-data>
   </default>
   <class>
       <data>class level</data>
   </class>
   <constructor expr="POJOConstructorTest()">
       <some-data>empty</some-data>
   </constructor>
   <method expr="void another(int, int)">
       <other-data>half</other-data>
   </method>
   <field name="somefield">
       <other-data>full</other-data>
   </field>
</metadata>
```

# Meta-Dados

- É possível definir o seu ClassMetadataLoader:

```
<metadata-loader tag="security"
  class="org.jboss.aspects.security.Sec
  urityClassMetaDataLoader"/>
```

- Uso:

```
<metadata tag="security" class="myPackage.*>
 qualquer coisa que eu quiser
</metadata>
```

# Próximo Seminário

- O que é Programação Orientada a Aspectos Dinâmica
- Como ela funciona no JBoss AOP
- O que é HotSwap
- O que é JVMTI:
- A POA Dinâmica implementada no JBoss AOP através do HotSwap:

# Referências

- JBoss AOP:
  - http://www.jboss.org/products/aop