POA Dinâmica

A Implementação de POA Dinâmica no JBoss AOP

POA Dinâmica

- Programação Orientada a Aspectos Dinâmica permite a adição e remoção de aspectos em tempo de execução.
  - Útil em sistemas grandes, em que é preciso adicionar ou remover uma funcionalidade sem que ocorra queda no servidor.
- Mas como funciona no JBoss AOP?

Instrumentação de Joinpoints

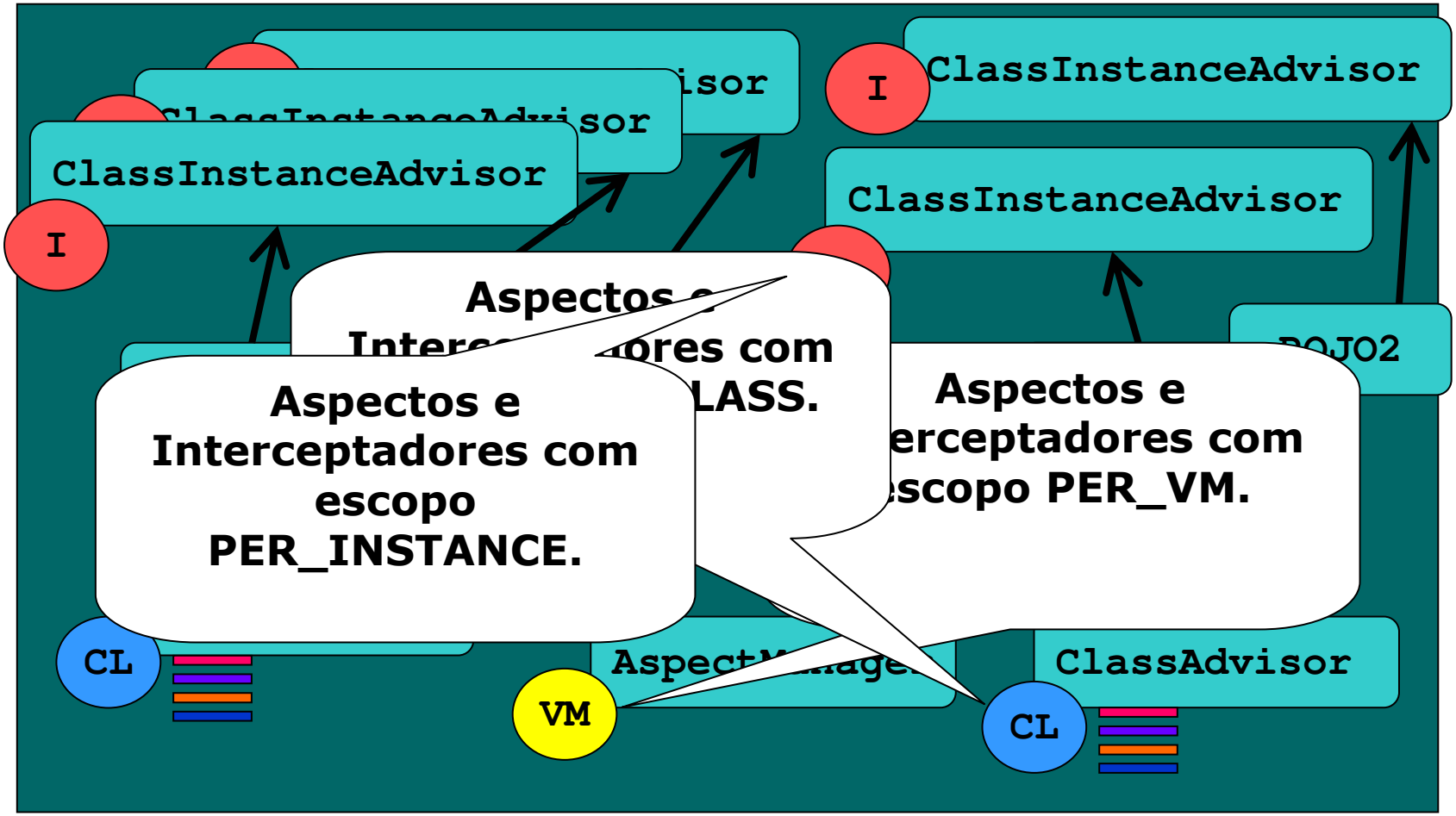
```
public class MyClass {
 public MyClass()
 {
   System.out.println("Hi");
 }
public static MyClass
       MyClass$aop$()
{ Invocation i = ...;
  i.invokeNext();}
}
```

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

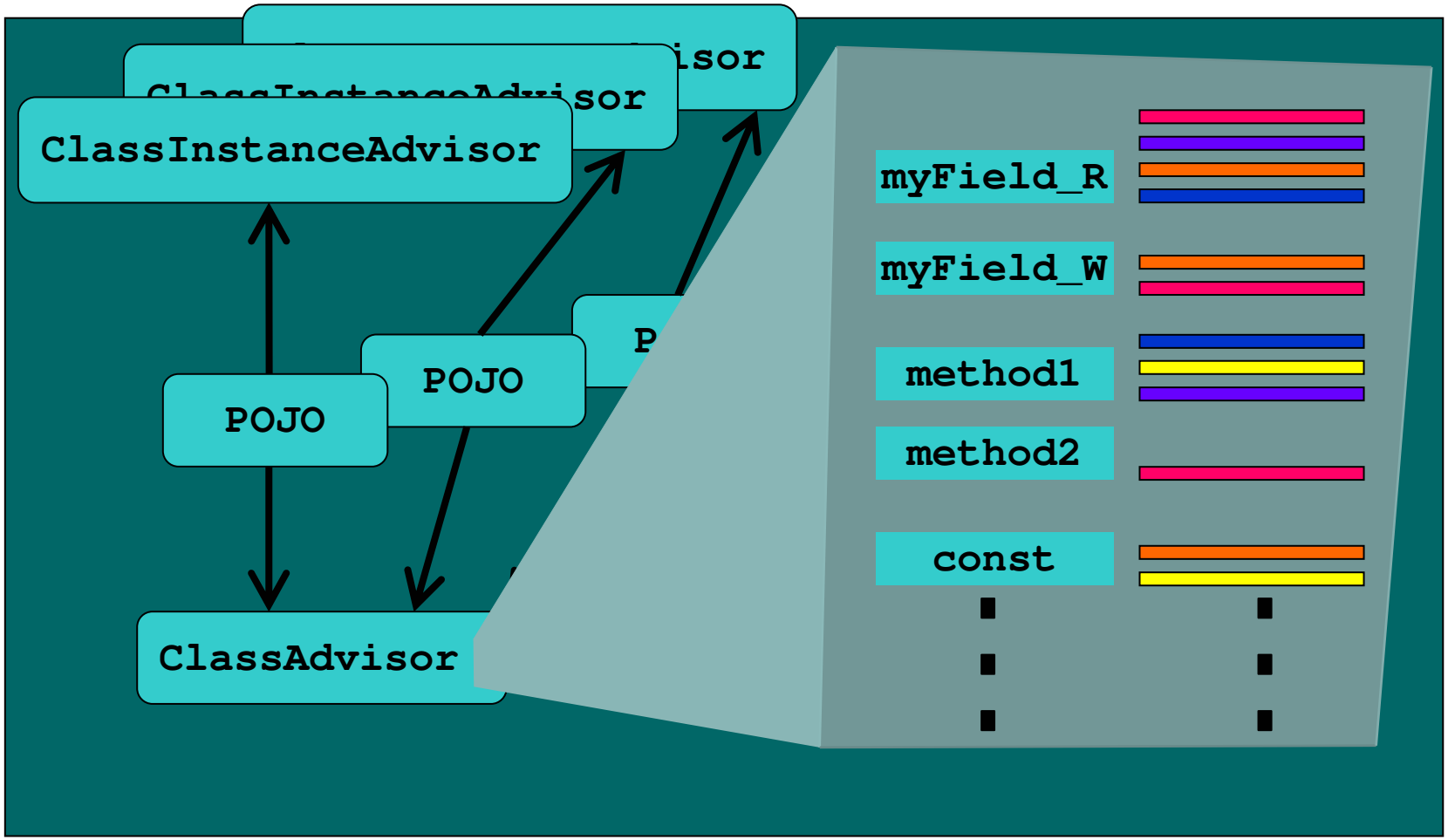
Instrumentação de Joinpoints

- Uma Invocation é criada e a pilha de interceptadores é invocada através dela (invokeNext()).
- Qual pilha de interceptadores?
  - O ClassAdvisor contém a pilha de interceptadores a ser utilizada.

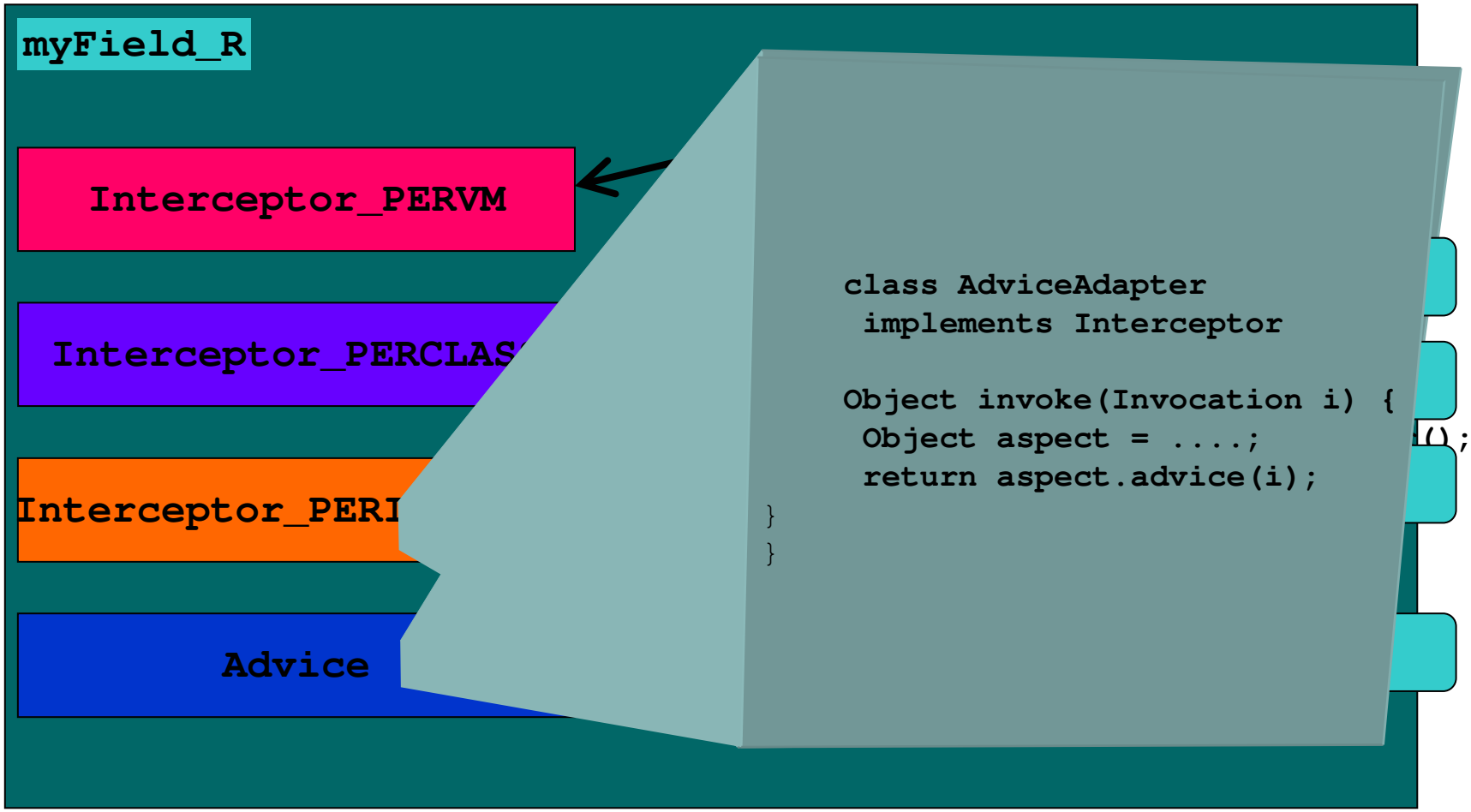
Class Advisors



ClassAdvisor



Pilhas de Interceptadores



Quais pontos são instrumentados?

- Todos os pontos de uma classe que são identificados por um pointcut:
  - Pointcut.matches
- Pointcuts registrados no AspectManager através de:
  - Declaração de um pointcut
  - Pointcuts plugáveis
  - Bindings

POA Dinâmica no JBoss AOP

- Possibilitar a adição e remoção *bindings* em tempo de execução.
- Atualmente isso é feito através da preparação de *pointcuts*.

Preparação de Pointcuts

- `<prepare expr="all(POJO)"/>`
-
```
@Prepare
public class POJO {
    ...
}
```
- Resulta na adição de um Pointcut com a expressão contida no prepare.

Preparação de Pointcuts

- Todos os *joinpoints* identificados pela expressão do prepare serão instrumentados da mesma forma que os identificados por *bindings*.
- Isso gera wrappers para pontos sem interceptadores;
- Dessa forma, quando um interceptador tiver que ser adicionado em tempo de execução, basta adicioná-lo na pilha de interceptadores adequada.

Operações de POA Dinâmica

- AspectManager:
  - addBinding(AdviceBinding) e removeBinding(AdviceBinding)
- Bindings são adicionados no sistema.
- Exemplo:

```
String p = "execution(POJO->new(..))“;
AdviceBinding b = new AdviceBinding(p, null);
binding.addInterceptor(MyInterceptor.class);
AspectManager.instance().addBinding(b);
```

Operações de POA Dinâmica

- Podemos também, adicionar interceptadores a uma instância:
  - InstanceAdvisor:
    - appendInterceptor(Interceptor i)
    - appendInterceptorStack(String stackName)
    - insertInterceptor(Interceptor i)
    - insertInterceptorStack(String stackName)
    - removeInterceptor(String interceptorName)
    - removeInterceptorStack(String stackName)
  - O interceptador é executado nos pontos:
    - Escrita e leitura de campos
    - Execução de métodos e de construtores

Operações de POA Dinâmica

- Exemplo de interceptadores por instância:

```
Interceptor i = new InstanceInterceptor();
Advised advised = (Advised)pojo;
InstanceAdvisor ia =
advised._getInstanceAdvisor();
ia.insertInterceptor(i);
```

O código wrapper

- Relembrando:

```
wrapper() {
 //obtém a pilha de ClassAdvisor
   Invocation i = ...;
   return i.invokeNext();
}
```

O código wrapper

- Claro que não queremos criar uma Invocation quando não existem interceptadores:

```
wrapper() {
if ClassAdvisor.interceptors is not empty OR
   target.instanceAdvisor.interceptor is not
   empty
do
   Invocation i = ....
   i.invokeNext();
else do execute joinpoint
}
```

Problemas com a Preparação

- Queda na performance desnecessária quando não existem interceptadores.
  - Exemplo:
    - Quero preparar o meu sistema (todos os *joinpoints*) para a ativação do log em tempo de execução, o que me permite analisar *bugs* encontrados sem desligar o sistema.
    - Quero poder fazer isso com o sistema em produção.

Solução: HotSwap

- HotSwap é a troca de bytecodes de uma classe em tempo de execução:
  - Possível através da JVMTI (Java Virtual Machine Tool Interface):um agente Java pode redefinir uma classe.
- Permite que mantenhamos o fluxo de controle intacto em pontos preparados enquanto não houver interceptadores.

Agentes de JVMTI

- Têm que implementar o método:

```
public static void premain(String agentArgs,
   Instrumentation inst);
```

- Com o Instrumentation é possível:
  - Registrar um ClassFileTransformer:
    `addTransformer(ClassFileTransformer cft);`
  - Redefinir (Hot Swap) classes através de:
    `redefineClasses(ClassDefinition[] def);`

Agentes de JVMTI no JBoss AOP

- Atualmente, é possível executar o JBoss AOP utilizando um agente, ao invés do compilador aopc:

```
java ... -javaagent:jboss-aop-jdk50.jar
```

- Porém, essa forma de utilização funciona de forma similar ao aopc:
  - Classes são instrumentadas antes de serem carregadas, sem o uso de hot swap.

Uma nova implementação

- Foi feita uma nova implementação da POA dinâmica para o uso de HotSwap com agentes da JVMTI.
- Diversas alterações no JBoss AOP tiveram que ser feitas, o que veremos a seguir.
  - O objetivo das alterações: garantir que um joinpoint não terá o seu fluxo de execução alterado quando não houverem interceptadores.

A transformação

- Divisão da transformação em duas etapas:
  - Preparação
    - Prepara um *joinpoint* para ser *wrapped* em um momento futuro (alteração de assinatura da classe que contém o joinpoint)
  - Wrapping
    - Instrumentação do joinpoint.

Diferenciação de Pointcuts

- Pointcuts são adicionados quando o usuário declara um:
  - Binding;
  - Pointcut;
  - Pointcut plugável;
  - Prepare;
- Diferenciar os *pointcuts* adicionados por *bindings* dos demais.

Classificação de Joinpoints

- Antes:

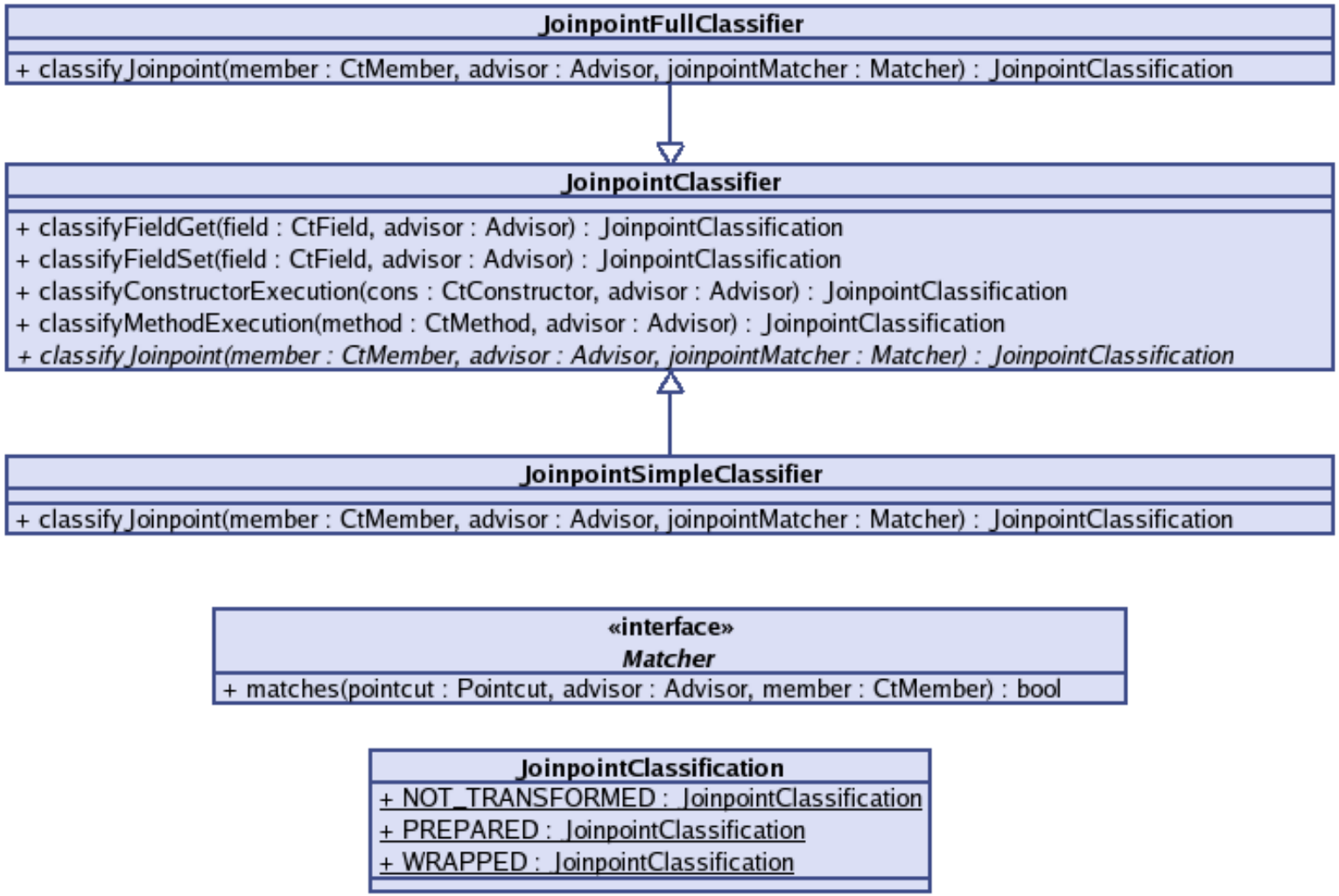
```
para cada pointcut
faça se pointcut.matches(joinpoint)
      então retorna WRAPPED;
retorna NOT_INSTRUMENTED;
```

Classificação de Joinpoints

- Agora:

```
Classificação = NOT_INSTRUMENTED;
Para cada pointcut
faça se classificação == PREPARED E
     pointcut.binding == null
          continue;
     se pointcut.matches(joinpoint)
     faça se pointcut.binding == null
          então classificação = PREPARED
          senão classificação = WRAPPED;
               break;
retorna classificação
```

Joinpoint Classification Design



Operação Dinâmica

- O que ocorre quando uma operação dinâmica é invocada?
  - **1º** Preciso verificar se alguns pontos do código passam a ser interceptados ou deixam de ser.

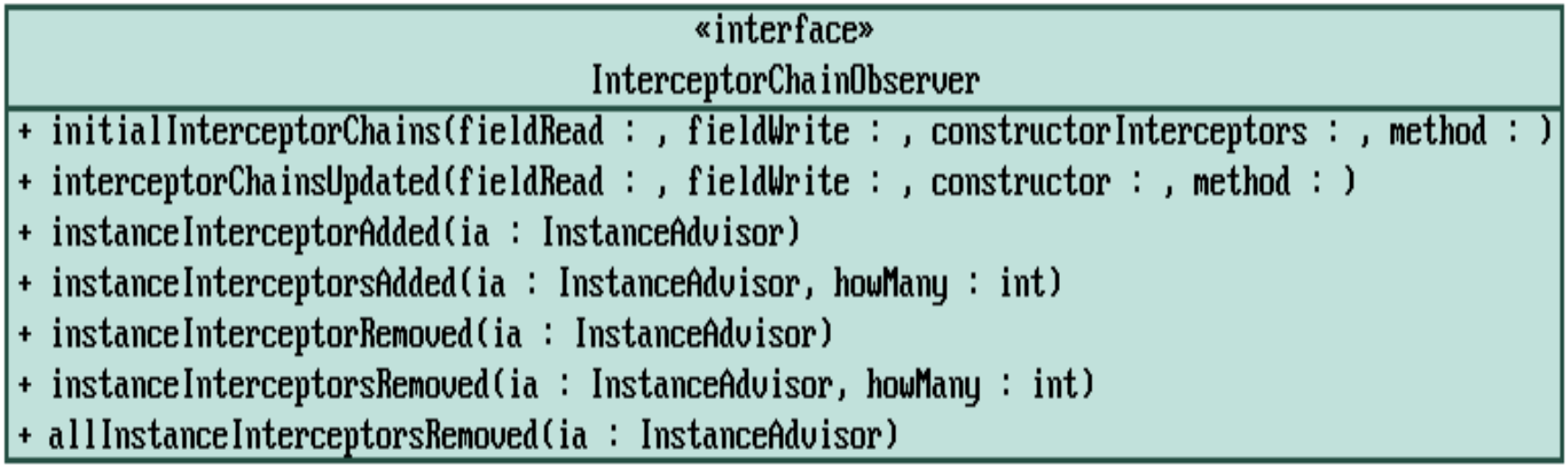
Pilhas de Interceptadores

- São alteradas através de operações de POA dinâmica;
- É preciso saber quando uma pilha:
  - se torna vazia: unwrap joinpoint
  - deixa de ser vazia: wrap joinpoint
- Nova interface:
  - InterceptorChainObserver
  - Observa o ClassAdvisor e ClassInstanceAdvisors

InterceptorChainObserver

- Existe um observador por classe.
- Está acoplado a:
  - o ClassAdvisor.
  - Os ClassInstanceAdvisors
    - Alteração do código do joinpoint em todas as instâncias devido a uma instância:
      - Um joinpoint é wrapped devido à adição de um interceptador ao ClassInstanceAdvisor.
      - E depois é unwrapped devido ao recolhimento da instância pelo coletor de lixo.

Interceptor Chain Observer



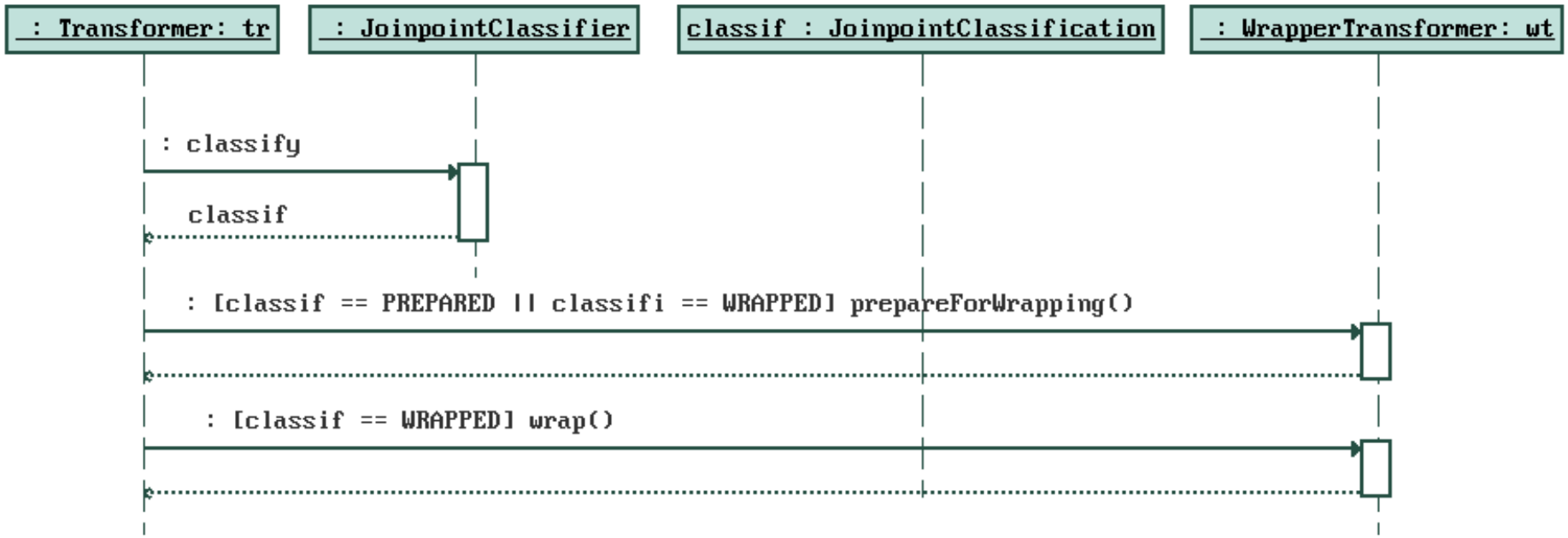
Operação Dinâmica

- O que é preciso fazer quando uma operação dinâmica é invocada?
  - **1º Verificar se alguns pontos do código passam a ser interceptados ou deixam de ser.**
  - **2º** Disponibilizar métodos wrap e unwrap nos transformadores.

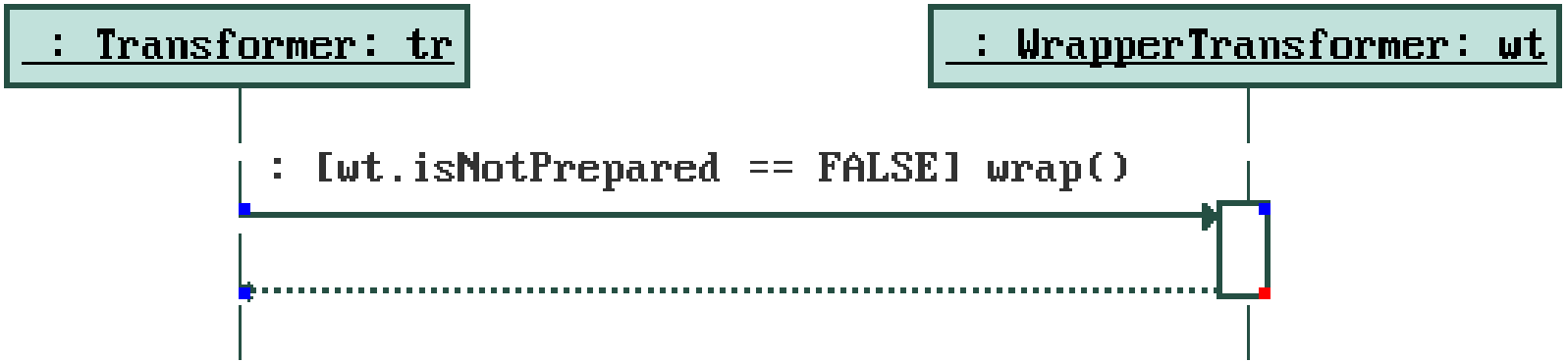
Operações de wrap e unwrap

- É preciso saber se o estado do joinpoint:
  - Não podemos mexer em joinpoints que não foram alterados.
  - Estados: NOT_PREPARED, WRAPPED ou UNWRAPPED
- Para isso, utilizamos um atributo associado ao componente da classe, indicando o estado de um pointcut.

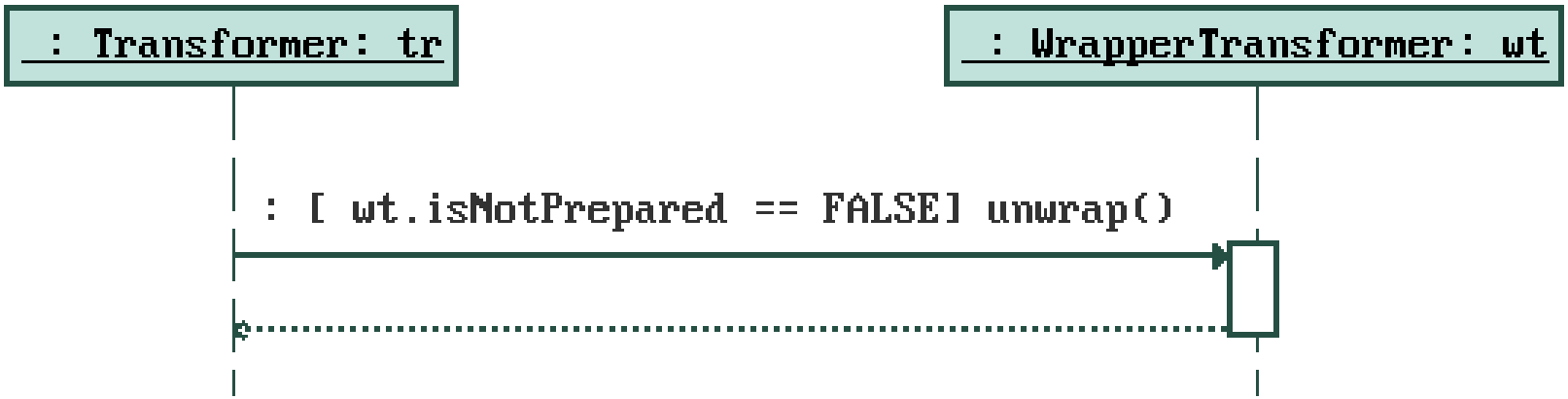
Transformer.transfrom()



Transformer.wrap()



Transformer.unwrap()



Operação Dinâmica

- O que é preciso fazer quando uma operação dinâmica é invocada?
  - **1º Verificar se alguns pontos do código passam a ser interceptados ou deixam de ser.**
  - **2º Disponibilizar métodos wrap e unwrap nos transformadores.**
  - **3º** Disponibilizar uma operação para alterar os *bytecodes* depois de uma operação dinâmica.

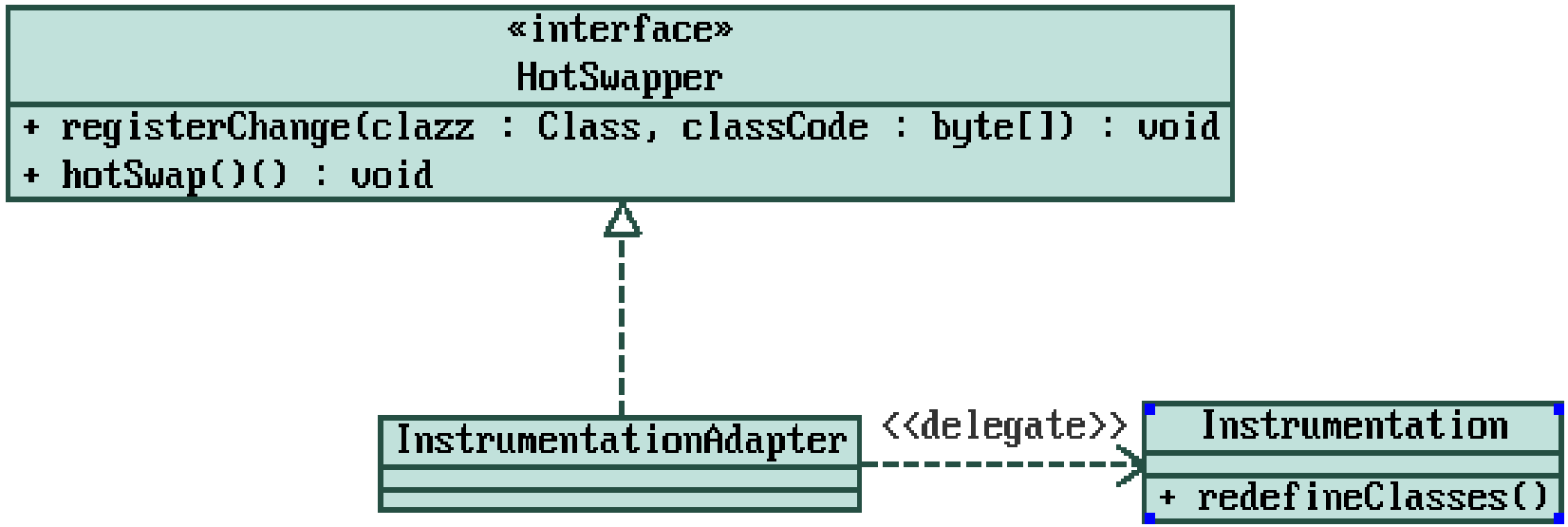
A fachada Instrumentor

- Uma vez executada uma operação de POA dinâmica, muitos pontos do código podem ter que ser wrapped ou unwrapped.
- Método adicionado a Instrumentor:
  - interceptorChainsUpdated.
    - Altera os bytecodes dos pontos que têm que ser wrapped ou unwrapped.

Redefinição de Bytecodes

- A redefinição dos bytecodes das classes alteradas tem que ser feita através de HotSwap.
- Solução flexível (suporta várias formas de HotSwap)
  - A interface HotSwapper
  - Para o uso com a JVMTI, o InstrumentationAdapter

HotSwapper


«interface»
HotSwapper
+ registerChange(clazz : Class, classCode : byte[]) : void
+ hotSwap()() : void
InstrumentationAdapter
<<delegate>>
Instrumentation
+ redefineClasses()

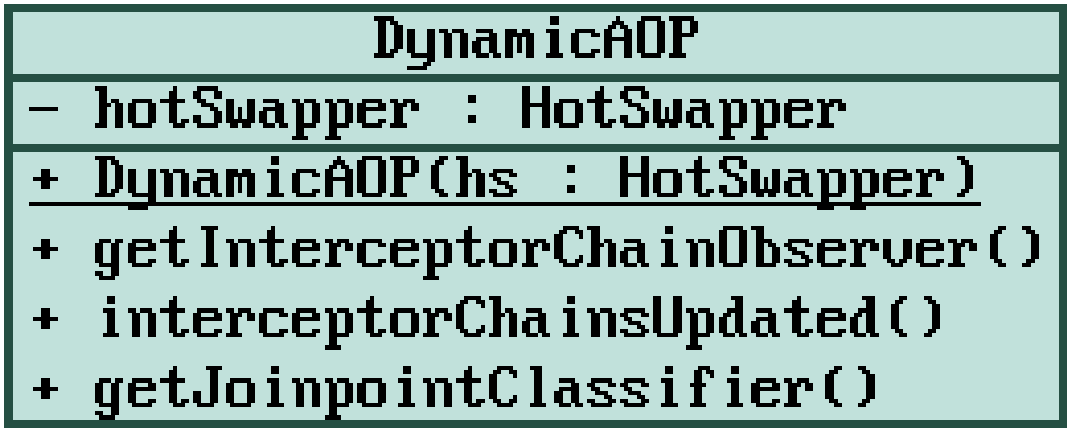
Operação Dinâmica

- O que é preciso fazer quando uma operação dinâmica é invocada?
  - **1º Verificar se alguns pontos do código passam a ser interceptados ou deixam de ser.**
  - **2º Disponibilizar métodos wrap e unwrap nos transformadores.**
  - **3º Disponibilizar uma operação para alterar os *bytecodes* depois de uma operação dinâmica.**
  - **4º** Alguém que gerencie essas funcionalidades.

DynamicAOP

- Fornece um InterceptorChainObserver para o ClassAdvisor;
- É notificada pelo AspectManager da execução de uma operação de POA dinâmica:
  - Quando isso ocorre, obtém do observer os JoinpointsStatusUpdate e invoca o método interceptorChainsUpdated passado um HotSwapper.
- Indica para o Instrumentor que o algoritmo de classificação de joinpoints a ser utilizado é o JoinpointFullClassifier.

DynamicAOP


DynamicAOP
- hotSwapper : HotSwapper
+ DynamicAOP(hs : HotSwapper)
+ getInterceptorChainObserver()
+ interceptorChainsUpdated()
+ getJoinpointClassifier()

Operação Dinâmica com HotSwap

- O que é preciso fazer quando uma operação dinâmica é invocada?
  - **1º Verificar se alguns pontos do código passam a ser interceptados ou deixam de ser.**
  - **2º Disponibilizar métodos wrap e unwrap nos transformadores.**
  - **3º Disponibilizar uma operação para alterar os *bytecodes* depois de uma operação dinâmica.**
  - **4º Alguém que gerencie essas funcionalidades.**
  - **5º** Isso tudo tem que ser plugável.

O uso de HotSwap

- Com HotSwap eu quero:
  - Usar JoinpointFullClassifier
  - Um InterceptorChainObserver que invoque operações no Instrumentor, informando de joinpoints que têm que ser wrapped ou unwrapped;
  - Fornecer uma implementação de HotSwapper para o Instrumentor;

O uso de HotSwap

- Sem HotSwap eu quero:
  - Usar JoinpointSimpleClassifier
  - Não quero um InterceptorChainObserver;
  - Não preciso fornecer HotSwapper para a execução de operações de wrap e unwrap, porque elas nunca serão chamadas.

DynamicAOPStrategy

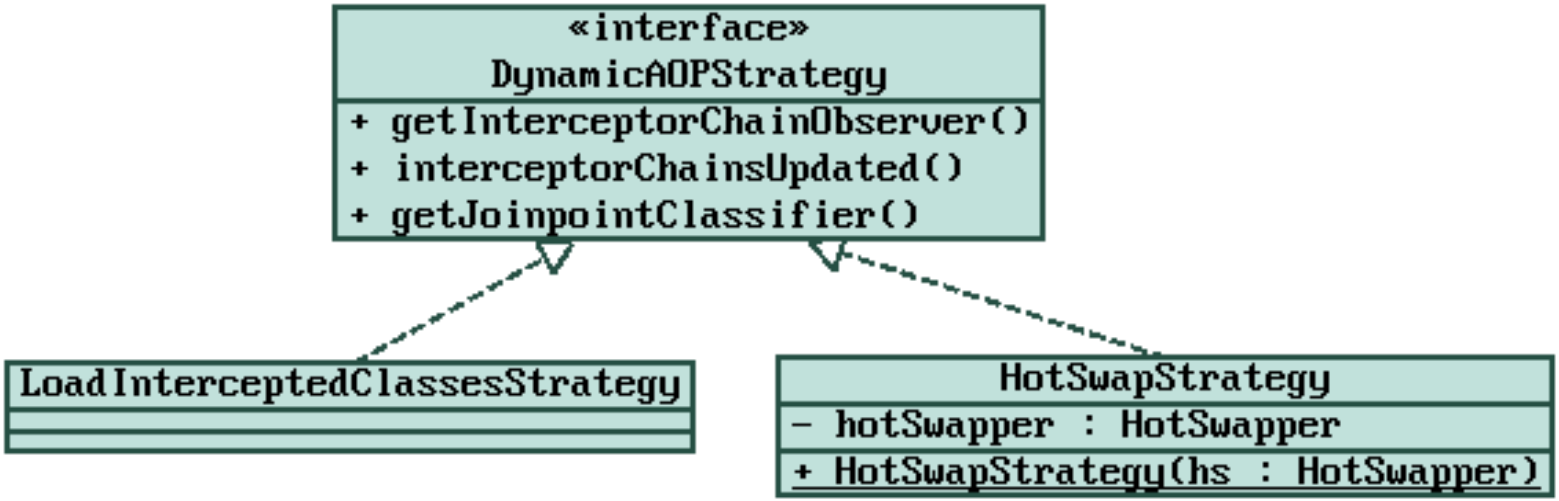
- Estratégia que indica comportamento diferente conforme a disponibilidade do HotSwap
  - LoadInterceptedStrategy
    - default.
  - HotSwapStrategy
    - Pode ser definida através do método AspectManager.XXX

DynamicAOPStrategy

- O agente Java 5 org.jboss.aop.standalone.Agent é responsável por executar:

```
AspectManager am = AspectManager.instance();
InstrumentationAdaptor hs = new
   InstrumentorAdaptor(instrumentation);
DynamicAOPStrategy ds = new
   HotSwapStrategy(hs);
am.setDynamicAOPStrategy(ds);
```

DynamicAOPStrategy - Arquitetura



Conclusão

- Melhor eficiência quando HotSwap estiver disponível;
- Pontos de extensão para outras soluções de HotSwap que não a JVMTI;
- O código do JBoss AOP se manteve compatível com a versão do Java 4.

Referências

- JBoss AOP:
  - http://www.jboss.org/products/aop
- JVMTI:
  - http://java.sun.com/j2se/1.5.0/docs/guide/jvmti/
  - http://java.sun.com/j2se/1.5.0/docs/api/java/lang/instrument/package-summary.html

Dúvidas?!?