A Arquitetura de Plugins no Eclipse

Alexandre Freire [alex@arca.ime.usp.br]

IME/USP

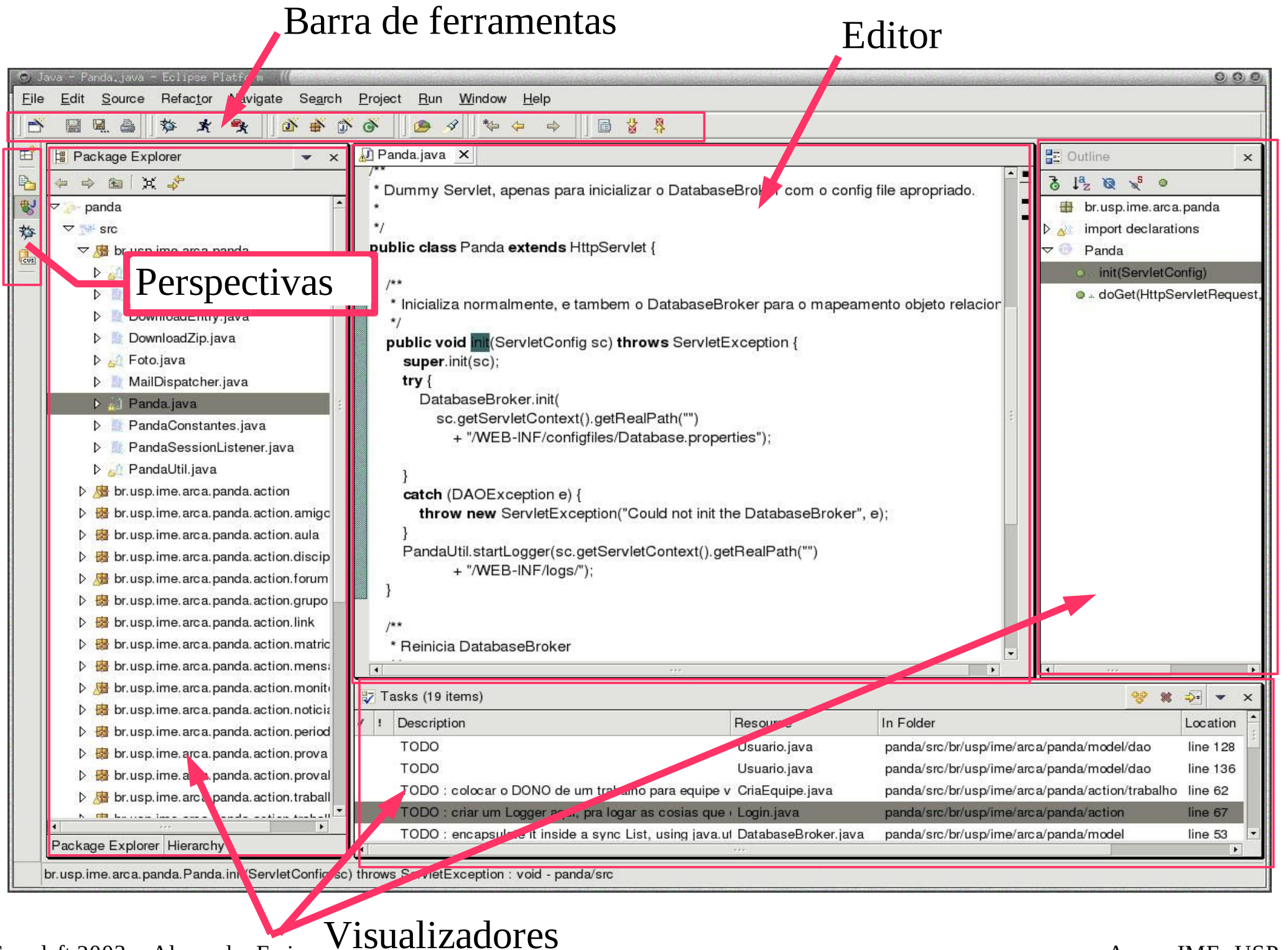
Oque é o Eclipse afinal?

Um IDE comum para todos (ou para o programador esquizofrênico...)

- O Eclipse não é uma IDE propriamente dita, é um arcabouço para desenvolvimento de ferramentas
- Estensível, aberto e portável. Através de *plugins*, diversas ferramentas podem ser combinadas criando um ambiente de desenvolvimento integrado
- Uma mesma plataforma para vários papéis de desenvolvimento: programador de componentes, integrador, responsável por testes, web designer...
  - O time inteiro usa a mesma aplicação
- Suporte para desenvolvimento de novas ferramentas
- O próprio Eclipse é desenvolvido usando o Eclipse

Arquitetura extensível

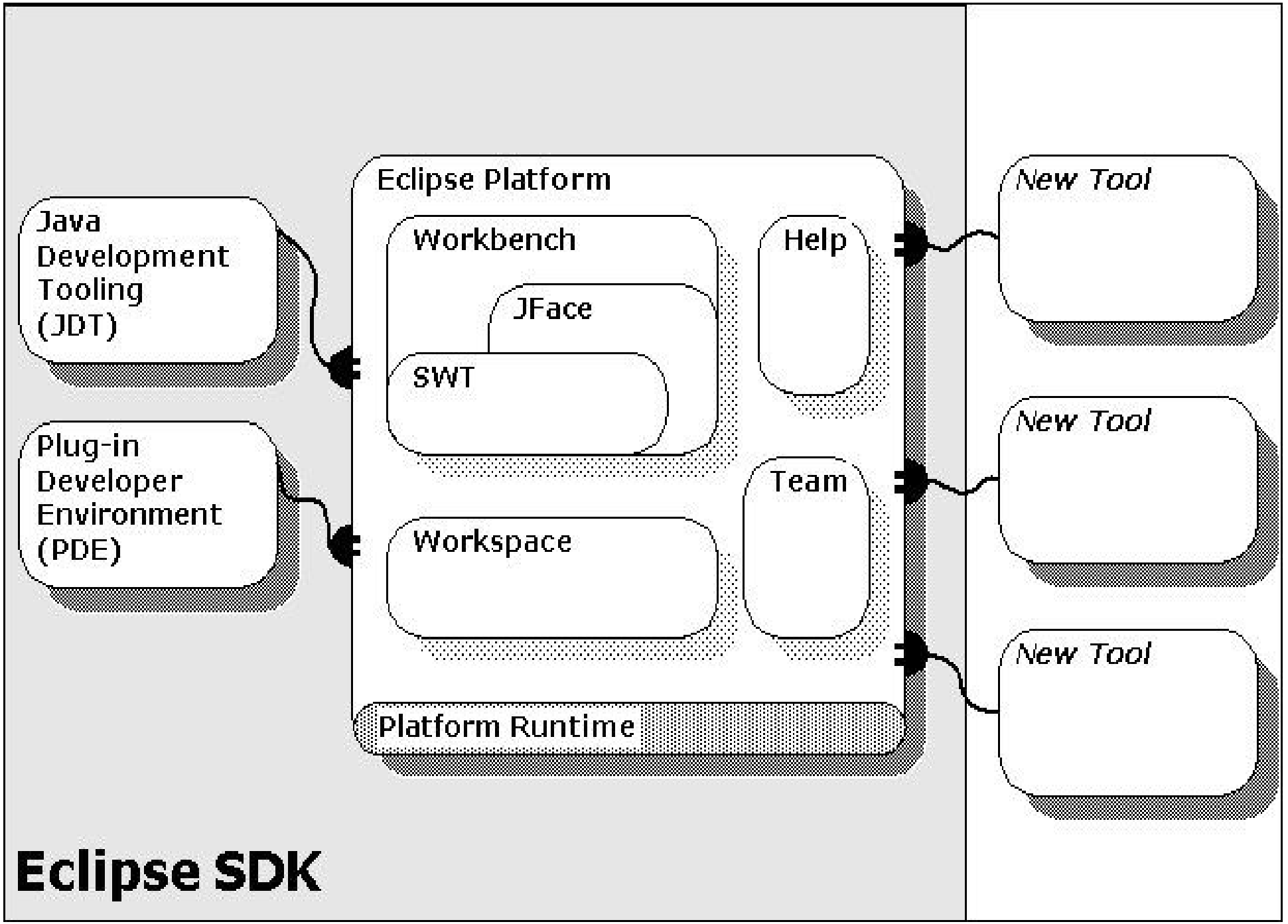
- Um plugin é a menor unidade extensível presente no Eclipse
  - pode conter código, recursos, ou ambos
- O próprio Eclipse é composto de diversos plugins
  - mais de 100 plugins
- Pontos de extensão
  - mecanismo que permite que um plugin adcione funcionalidade a outros plugins
- Interações de usuário são padronizadas
  - plugins de diferentes vendedores são integrados
  - reutilização de componentes – “IDE patterns”



A bancada de trabalho

O iluminismo nas IDEs, é tudo uma questão de perspectiva

- O Eclipse usa a abstração de bancada de trabalho para prover uma interface comun para seus usuários
- A bancada contém menus, barras de ferramentas, editores e visualizadores, estes componentes são extensíveis
- A bancada pode ter várias perspectivas, cada perspectiva organiza diferentes editores e visualizadores e contribui com menus e barras de ferramentas, determinando a aparência da bancada
- Visualizadores oferecem diversas informações sobre os recursos de um projeto. Podem apresentar somente partes ou atributos internos de um recurso ou podem ser ligadas a um editor.
- Perspectivas diferentes para desenvolvedores diferentes


Eclipse Platform
Java Development Tooling (JDT)
Plug-in Developer Environment (PDE)
Workbench
JFace
SWT
Help
Team
Workspace
Platform Runtime
New Tool
New Tool
New Tool
Eclipse SDK

Estrutura da plataforma

- Plataforma de Runtime
  - Define pontos de extensão e o modelo de plugins. Descobre plugins dinâmicamente e mantêm as informações em um registro.
- Workspace (gerência de recursos)
  - Define API para criar e gerenciar recursos (projetos, arquivos e pastas) produzidos por ferramentas e armazenados em disco
- Workbench e UI
  - Define interface comun para usuário lidar com recursos e ferramentas. Define pontos de extensão para adcionar, por exemplo, visualizadores ou ações no menu. Contém toolkits para implementação de interfaces (SWT e JFace)

Estrutura da plataforma - cont.

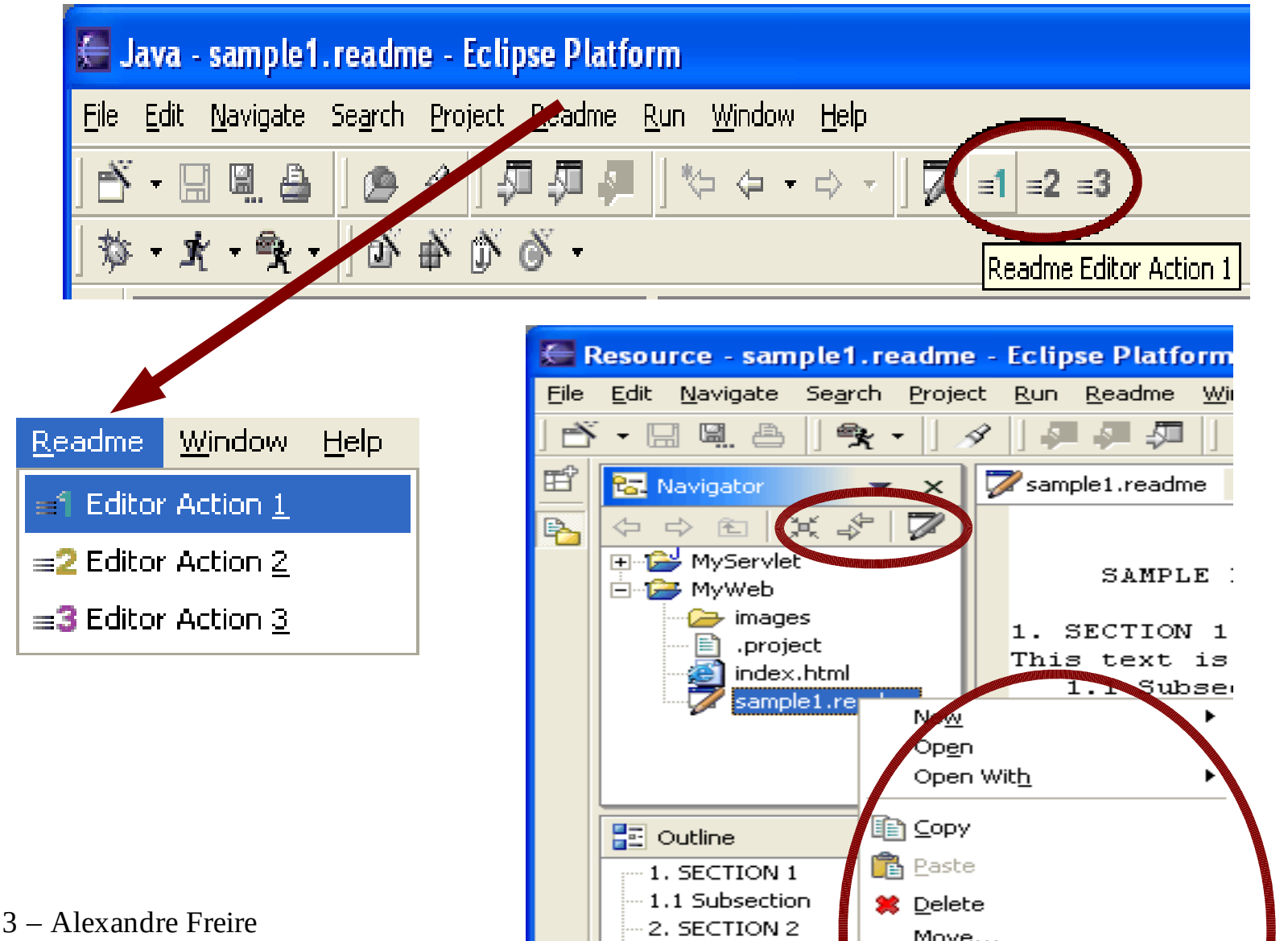
- Sistema de ajuda
  - Define pontos de extensão para plugins proverem ajuda ou documentação
- Suporte a times
  - Define um modelo de gerência e controle de versões dos recursos de um projeto para times
- JDT
  - Ferramentas para criação de um ambiente de desenvolvimento para Java. Conjunto de plugins que extendem a plataforma para visualizar, editar, compilar, depurar e rodar código em Java.
- PDE
  - Ambiente de desenvolvimento e execução de plugins. Define ferramentas que cuidam da criação, manipulação, depuração e implantação de plugins

Plugins são componentes

- Unidades de código reutilizáveis e estensíveis
- Normalmente representam um componente completo do ponto de vista do usuário final
- Definido em várias classes e pacotes
- Componentes com interface gráfica são normalmente divididos em dois plugins, uma para o modelo e outro para interface
- Plugins podem extender pontos de extensão de diversos outros plugins e definir seus próprios pontos de extensão
- São implantados no sub-diretório plugins do diretório de instalação do Eclipse e descobertos em tempo de execução

Pontos de extensão

- Ações
  - do workbench, de editores ou de visualizadores



Mais pontos de extensão

- Novos editores, visualizadores ou perspectivas
- Páginas de ajuda e preferências do plugin
- Wizards
  - criação de recursos
- Natures e Builders
  - compilar recursos
- Markers e Decorators
  - padronizar a visualização de recursos específicos do seu plugin
- Filtros de extensões
  - filtros podem indicar ou não a necessidade de adcionar uma extensão

Estrutura de um plugin

- $ECLIPSE_ROOT/plugins/br.ime.usp.arca.plugin_1.0.0/
  - plugin.xml -> Manifesto do plugin
  - plugin.properties -> i18n, etc...
  - about.html -> licensa
  - *.jar -> classes
  - /lib/*.jar -> + classes
  - /icons/*.gif ou *.png -> imagens
- Só a maior versão do plugin vai ser carregada
- Plugins só tem acesso a classes exportadas por outros plugins
- Cada plugin é carregado pelo seu próprio ClassLoader

Ciclo de vida de um plugin

- Durante a inicialização o Eclipse lê todos manifestos de plugins implantados no diretório
- As informações relativas a todos plugins encontrados são armazenadas em um registro de plugins que fica disponível para todos outros plugins em tempo de execução
- Os plugins não são carregados, permitindo que o tempo de abertura do Eclipse seja constante independente do número de plugins instalados, os plugins e suas respectivas classes só são carregados quando estritamente necessário (lazy-loading)
- Plugins ainda não podem ser descarregados

Exemplo de manifesto

```
<?xml version="1.0" encoding="UTF-8"?>
<plugin
   id="br.ime.usp.arca.plugin"
   name="Plugin Demo"
   version="1.0.0">
   <runtime>
      <library name="plugin.jar"/>
   </runtime>
   <requires>
      <import plugin="org.eclipse.ui"/>
   </requires>
</plugin>
```

Plugin ClassLoader

- O PluginClassLoader define quais classes são visíveis para cada plugin baseado nos tags <requires> e <runtime> do manifesto do plugin
- <runtime> define onde está o código e recursos do plugin, um atributo opcional do tag “library”, chamado “export” permite que outros plugins tenham acesso ao jar especificado
- <requires> define outros plugins necessários para carregar este plugin. Os plugins são referenciados por ID e não com referências diretas as suas bibliotecas
- Plugins não tem acesso ao classpath normal, por motivos de segurança (?), isto torna muito comun a criação de plugins que servem de “casca” ao redor de bibliotecas (eg, plugin do JUnit)

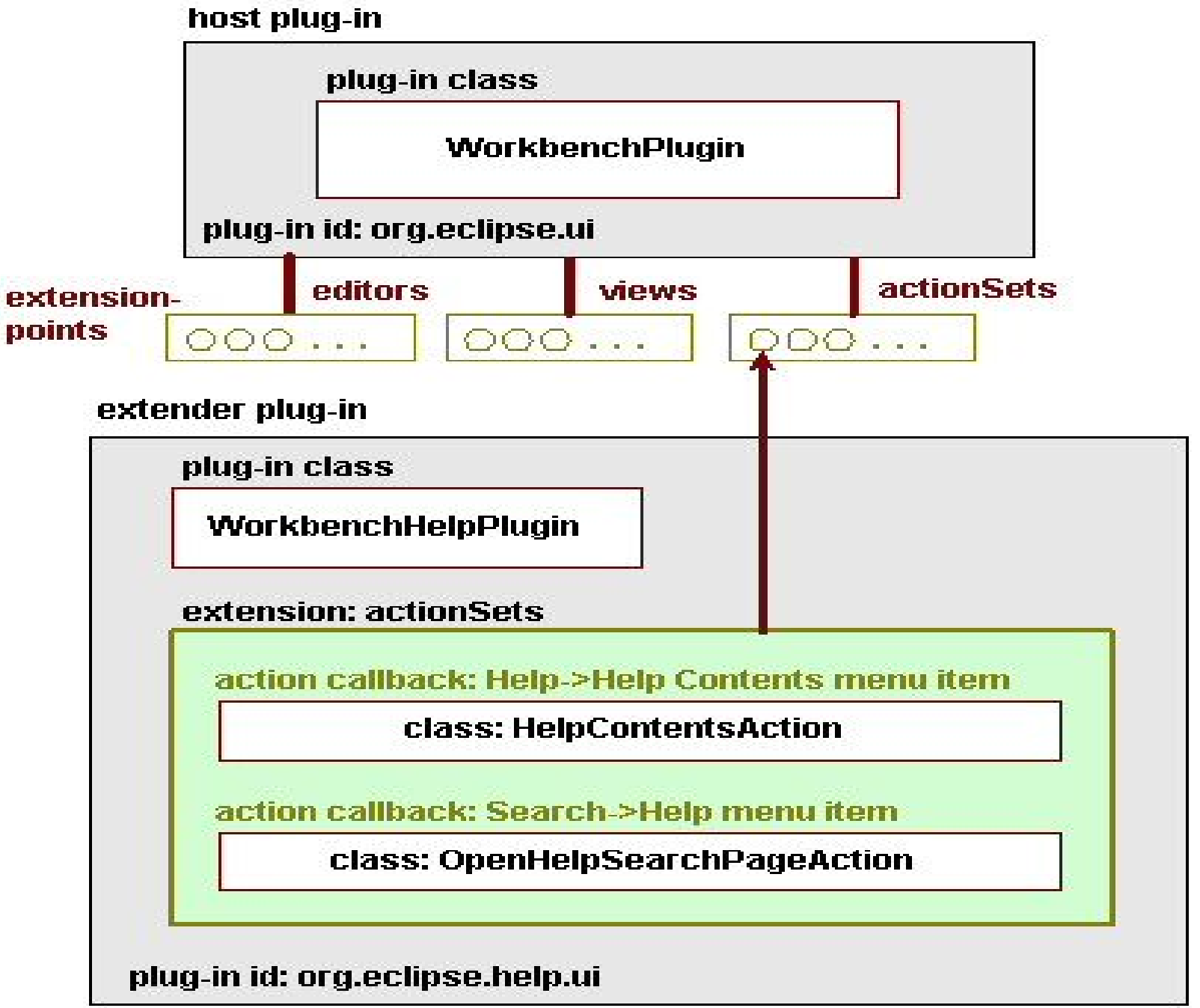
Extensões e inversão de controle

- A plataforma de responsabiliza por verificar quais pontos de extensão um plugin extende, se responsabilizando por construir e fazer chamadas as classes do plugin quando necessário

- Um plugin pode definir pontos de extensão (host plugin) no seu manifesto, ao fazer isto, normalmente define também um schema e uma interface de callback, que caracterizam a extensão e permitem que os plugins que o extendem sigam um mesmo padrão.

- O schema define elementos que são necessários para que o Eclipse possa criar componentes na sua interface sem necessariamente ter que carregar o plugin.

- Plugins extendem outros em seu manifesto (extender plugin) e definem propriedades da sua extensão



Tarefas do Host Plugin

- Define extension points
  - <extension-point id=”Action Sets” schema=”schema/actionSets.exsd”>
- Durante sua inicialização cria *Virtual Proxies* para cada plugin que o extende e cria componentes em sua interface que chamam estes proxies
- Ao ser chamado pela primeira vez, o Proxy delega suas tarefas para a classe definida pelo extender plugin, que somente neste momento é carregado pela plataforma
- Plugins podem ter pontos de extensão que não tem efeito em sua interface, neste caso é possível que um extender plugin nem precise definir classes para criar CallBack Objects
- Outro padrão utilizado para lazy-loading eh *Virtual Adapter*

Tarefas do Extender Plugin

```
<!-- Action Sets -->
    <extension
         point="org.eclipse.ui.actionSets">
        <actionSet
               label="Help"
               visible="true"
               id="org.eclipse.help.internal.ui.HelpActionSet">
         <action
               label="&Help Contents"
               icon="icons/view.gif"
               helpContextId="org.eclipse.help.ui.helpContentsMenu"
               tooltip="Open Help Contents"
               class="org.eclipse.help.ui.internal.HelpContentsAction"
               menubarPath="help/helpEnd"
               id="org.eclipse.help.internal.ui.HelpAction">
            </action>
            <!-- ... other actionSet elements -->
         <action
               label="&Help..."
               icon="icons/search_menu.gif"
               helpContextId="org.eclipse.help.ui.helpSearchMenu"
             class="org.eclipse.help.ui.internal.OpenHelpSearchPageAction"
               menubarPath="org.eclipse.search.menu/dialogGroup"
               id="org.eclipse.help.ui.OpenHelpSearchPage">
            </action>
        </actionSet>
    </extension>
```

Tarefas do Extender Plugin

- Define callback objects para cada extensão
- Pode extender mais de um extension point
- Pode extender mais de um host plugin
- Um único extension point pode ter mais de um callback object
- Define atributos necessários para que o host plugin modifique sua interface sem ter que carregar o plugin
- Pode prover seus próprios extension points
- Pode extender seus próprios extension points

Conclusões

- O Eclipse define uma arquitetura extensível baseada em componentes fracamente acoplados
- Plugins são componentes que caracterizam um sistema coeso sendo implantados em tempo de execução e carregados somente quando necessário
- Pontos de extensão criam uma arquitetura facilmente extensível que permite a criação de uma verdadeira rede de componentes

Muito obrigado!

arca.ime.usp.br

alex@arca.ime.usp.br