



# The Eclipse Rich Client Platform

Slides by various members of the  
Eclipse JDT and Platform teams



# Outline

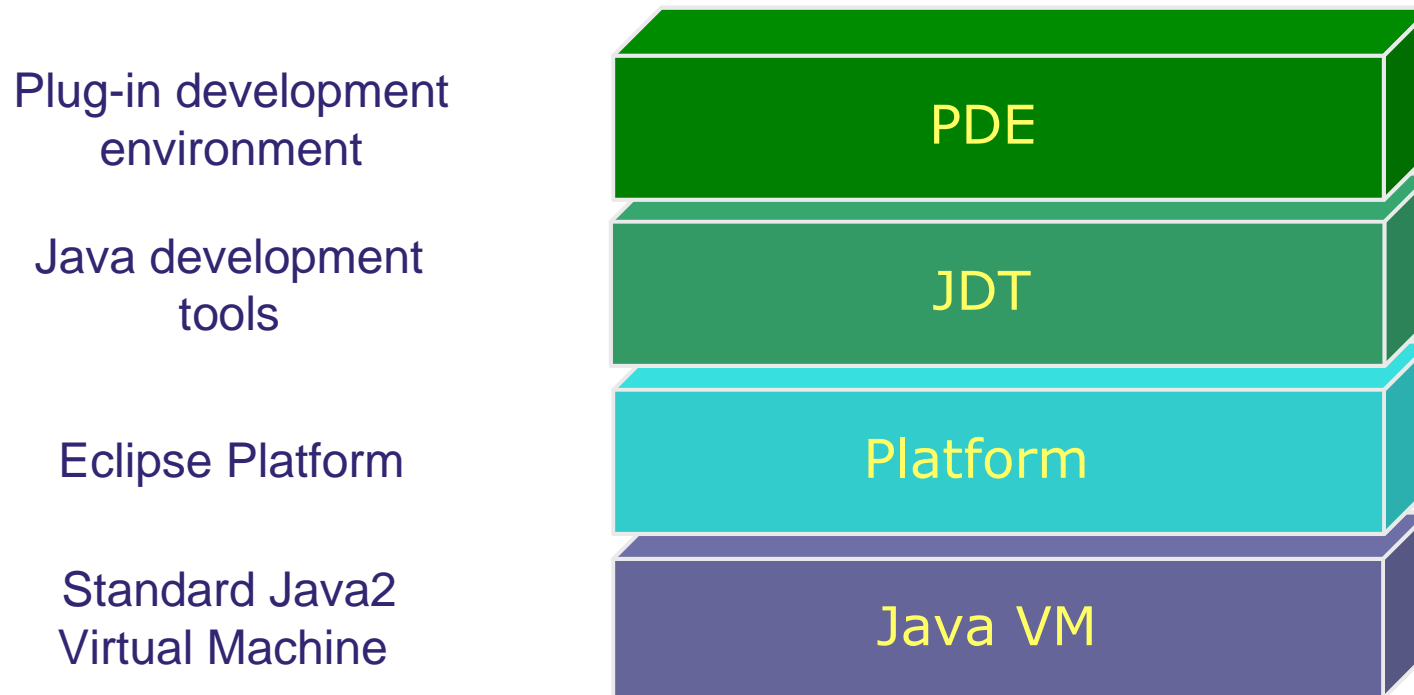
---

- Rich Client Application ?
- The Eclipse Plug-in Architecture
- Eclipse Plug-ins in action
- The Eclipse Rich Client standard components
- How to deploy Plug-ins
- How to develop Eclipse Plug-ins

# How Eclipse started

---

- Eclipse is a universal platform for integrating development tools
- Open, extensible architecture based on plug-ins





# Eclipse for Non IDE Applications

---

“could I dump all the plug-ins that come with eclipse and use the platform to host only business specific plug-ins that have been built?” – [news.eclipse.org](http://news.eclipse.org)

But when using Eclipse 2.1 as an ordinary application platform you get either:

➤ **too much**

- all the IDE specific components
- user interface is polluted with IDE specific actions

➤ **too little**

- only SWT and JFace
- low level programming model
- No extensibility



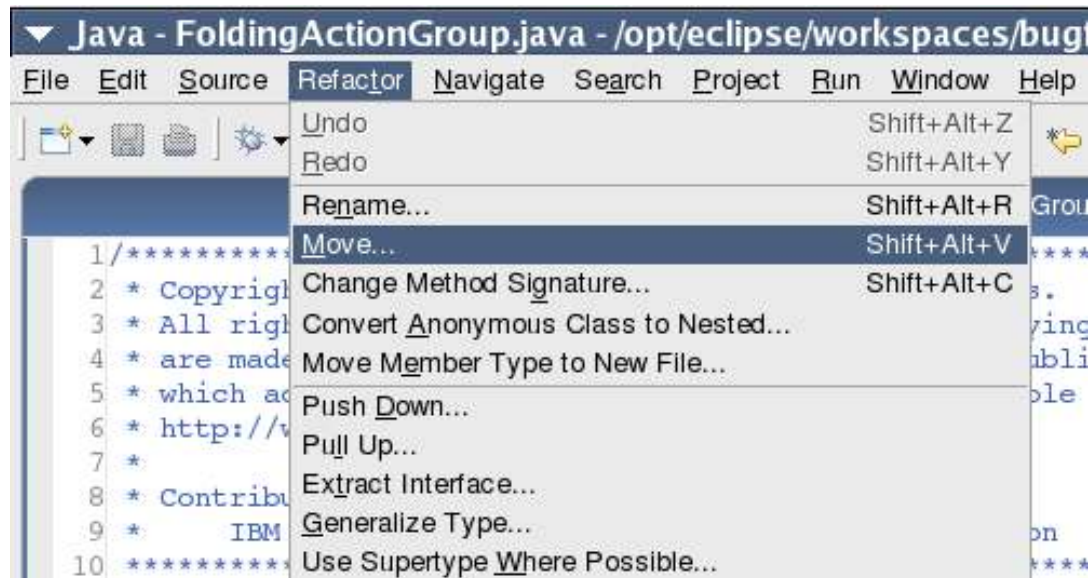
# Towards a Rich Client Platform

---

- **Many workbench components are not IDE specific. Advanced desktop applications have similar needs**
  - open architecture
  - efficient, configurable, portable user interface
  - supports product branding, install/update support
  - integrated help, user configuration/preferences
  
- **Enable workbench to be used for non IDE applications**
  - remove IDE personality from workbench
    - no built-in editors, views, perspectives
  - remove assumption that workspace is the data model
  - make most other components optional
    - rich function, low footprint

# What is a Rich Client Application?

- An application that use the windowing and GUI features of the operation system they run on. This means:
  - Native widgets, menu and tool bars
  - Drag & Drop
  - Integrates with platform component model
  - ...





## Consequences...

---

- ✓ More responsive user experience
- ✓ Better integration with existing Desktop tools
- ✓ Lower server loads
- ✓ Offline execution
- ✓ Local data access
- Client/Server architecture
- ⚠ Memory footprint
- ⚠ Management & Deployment



# Eclipse Rich Client Platform

---

A Rich Client Platform needs a strong component model with the following major characteristics:

- ✓ Specified interfaces: a component must declare its public API and how it can be extended
- ✓ Lazy loading: components are loaded on demand not on startup
- ✓ Versioning: prerequisite components are reference by name and version
- ✓ Dynamic detection: components are detected dynamically (no need to restart)

Additionally the following issues must be addressed:

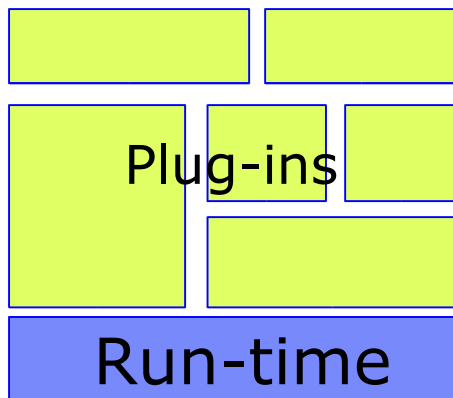
- ✓ Managing: install, update, remove & discover components
- ✓ Development: IDE to develop components
- ✓ Security: based on Java 2 security

# Platform vs. Extensible Application

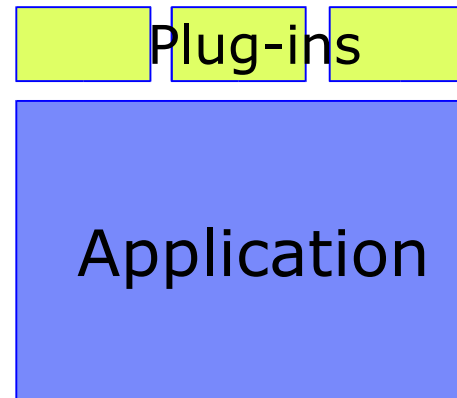
---

- Eclipse Rich Client Platform
  - It has an open, extensible architecture
  - Built out of layers of plug-ins

## Platform

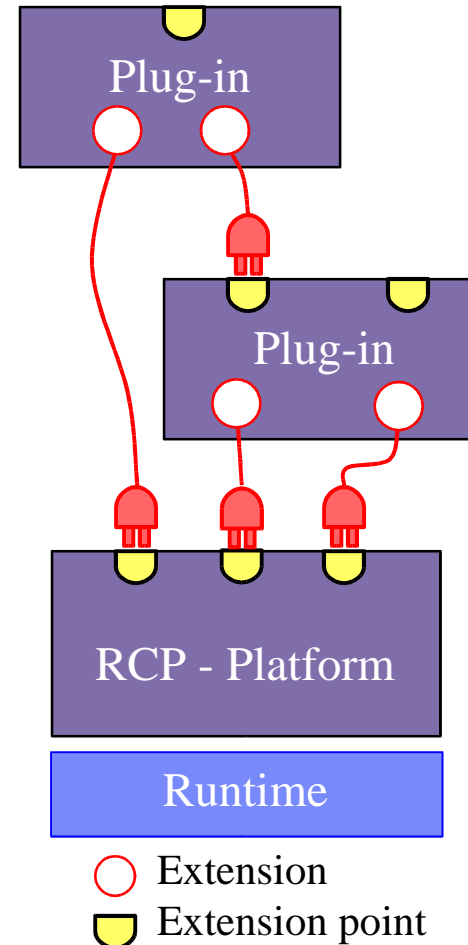


## Extensible App



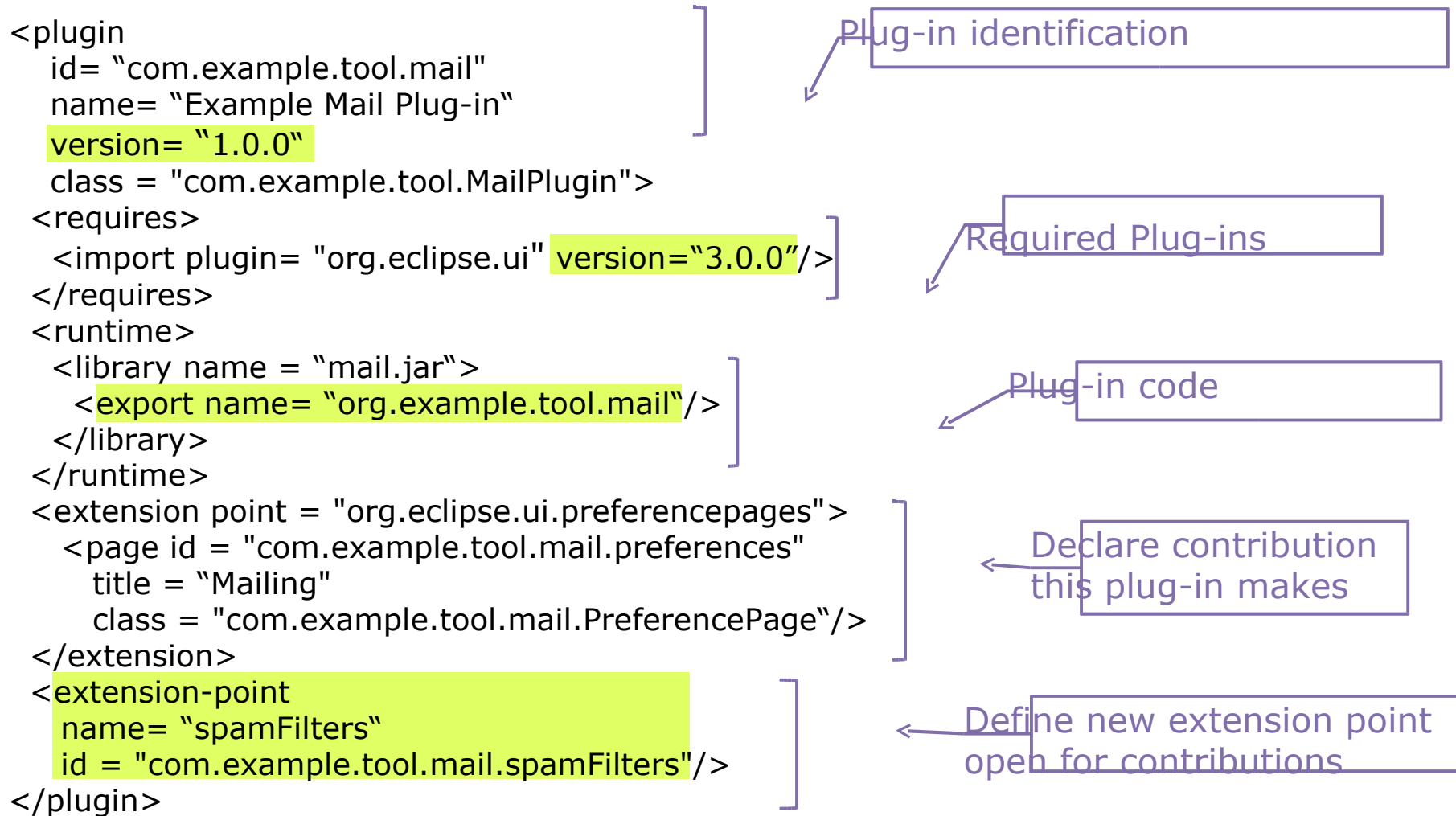
# Eclipse Plug-in Architecture

- **Plug-in == Component**
  - Set of contributions
  - Smallest unit of Eclipse function
  - Details spelled out in plug-in manifest
  - Big example: mail client
  - Small example: action to calculate the number of lines of a mail
- **Extension point** – named entity for collecting contributions
  - Example: extension point to add additional spam filtering tools
- **Extension** – a contribution
  - Example: a specific spam filter tool
- **RCP - Platform** – set of standard plug-ins
- **Runtime** – controls and manages contributions





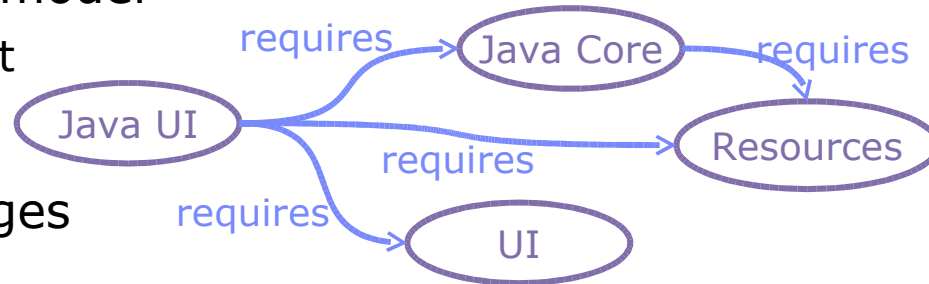
# The Plug-in Manifest



# The Eclipse Runtime

---

- Java component (*plug-in*) model
  - dependency management
  - activation management
- Extension registry - manages
  - extension points and
  - corresponding extensions
- OSGI based (Open Service Gateway Initiative):
  - Nokia, NTT, Motorola, Philips, Siemens, Oracle
  - dynamic install/uninstall/update of components
  - service architecture
  - security (based on Java 2)
  - remote configuration API



# Lazy Loading

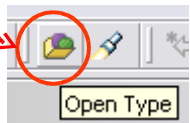
```
<action  
  toolbarPath="search"  
  tooltip="Open Type"  
  icon="icons/opentype.gif"  
  class="org.eclipse.jdt.OpenTypeAction"/>
```

Declarative  
Definition  
(manifest)

lazily instantiated  
using reflection

Procedural  
Implementation  
(Java JAR)

contribution  
implementation



# Hello World Example

---

A component based Hello World application that allows contributing additional greeters:

- Uses OSGI as a runtime
- Provides extension points and extensions
- Demonstrates how Eclipse technology can be used to componentize existing applications

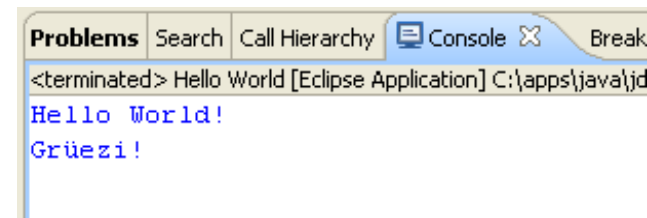
demo.helloworld.swiss

Swiss greeter

demo.helloworld

Extension point for greeter  
Default greeter for Hello World

Runtime (OSGi)



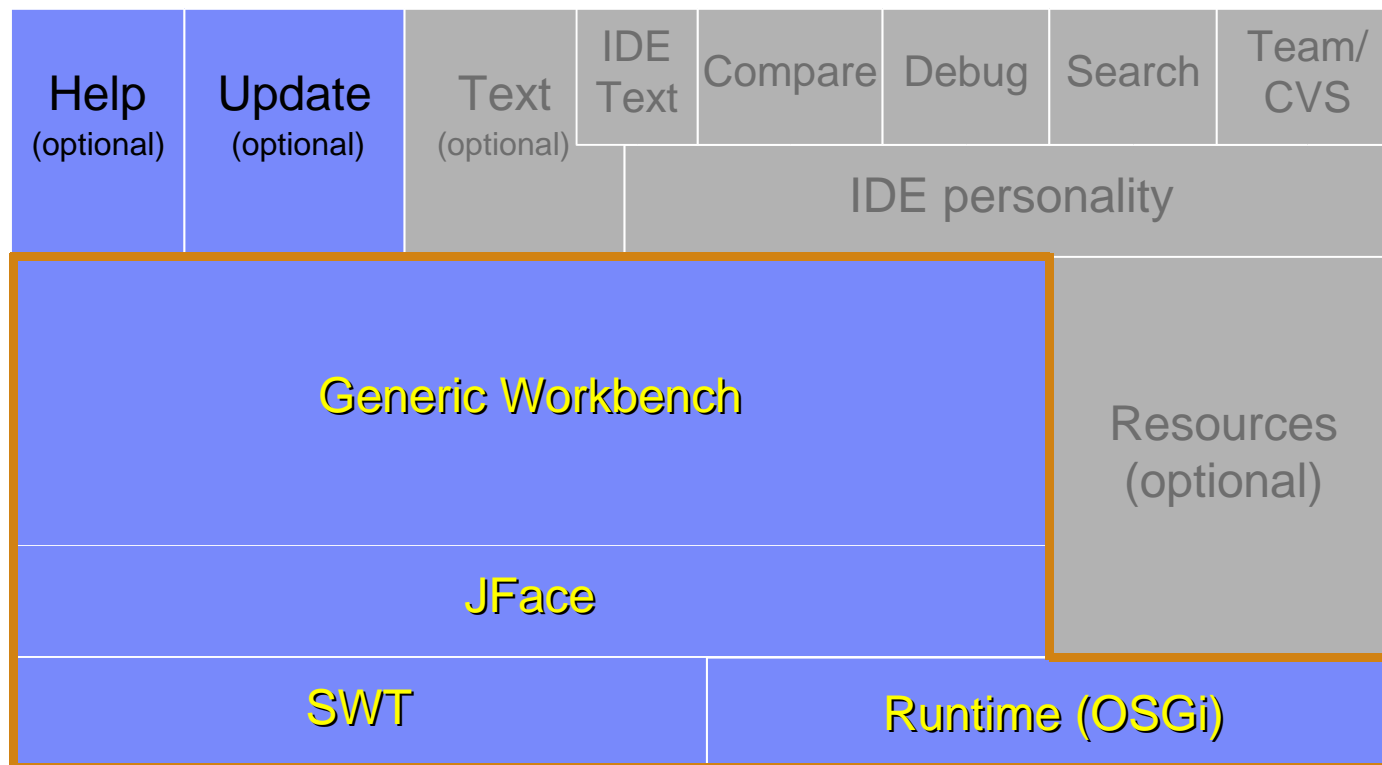
```

Problems Search Call Hierarchy Console Break
<terminated> Hello World [Eclipse Application] C:\apps\java\jd
Hello World!
Grüezi!
  
```



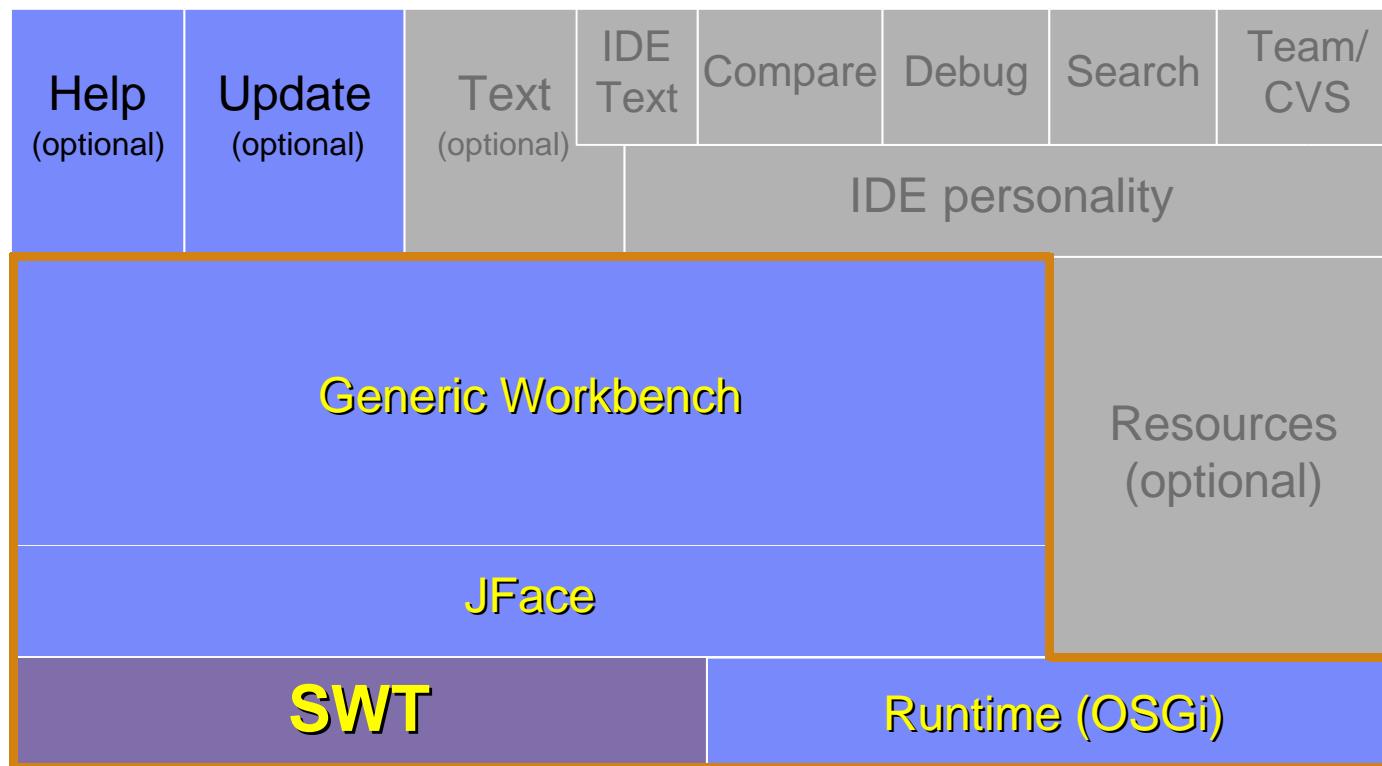
# Eclipse Platform – Version 3.0

---



# Standalone Applications

---





# Standalone Applications

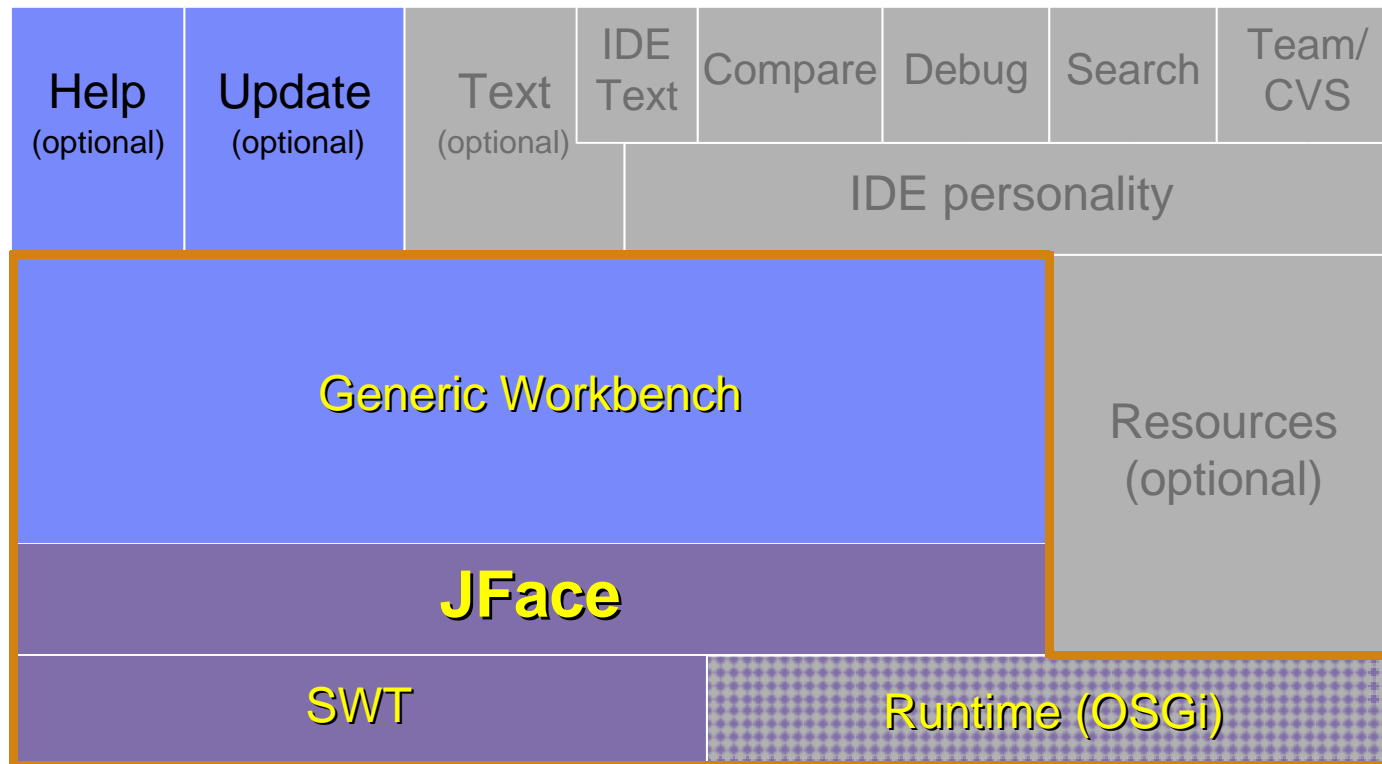
---

- Application model
  - Single not extensible application
- **Standard Widget Toolkit**
  - Platform independent widget toolkit
    - Native widgets (button, tree, table, menu, ...)
    - Win32, GTK, Motif and Mac
  - Integrates with other native application
  - Support for OS component model
    - OLE under Win32
- Programming model
  - OO widget library – no framework
  - API equivalent to native Win32 or GTK applications



# Extensible Applications

---





# Extensible Applications

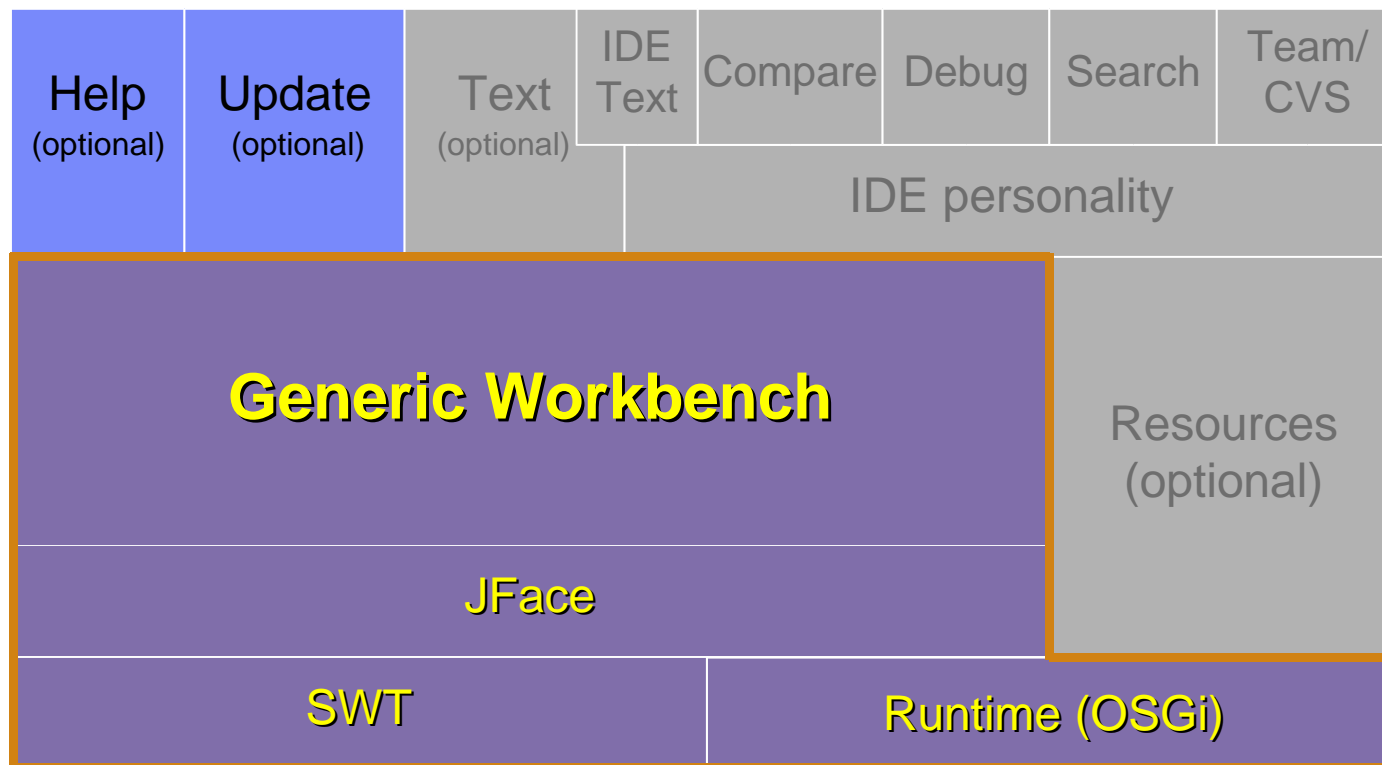
---

- Application model
  - Single application
- JFace – brings:
  - MVC concept: viewer & content provider
  - Application window: menu bar, tool bar, content area & status line
  - Action support: menu bar, toolbar, context menu
  - Preference and wizard framework
  - No extension points, API only
- Runtime – brings:
  - Change for extensibility
- Programming model
  - Formed by Model View Controller paradigm
  - „Frameworkish“



# Application Platform

---





# Application Platform

---

- Application model
  - Family of components (Mailing, Organizer, Address-Book, ...)
  - Different sets of components form different applications
- Workbench – brings:
  - Perspectives: define arrangement of editors and views
  - Editors: edit or browse a document or input object
  - Views: navigate a hierarchy of information
  - Action contributions: add additional action to already existing elements
  - Manages shared resources like global menu, preference pages, ...
- Programming model
  - Components contribute to workbench extension points
  - Components provide own extension points
  - Split between XML (plugin.xml) and Java code

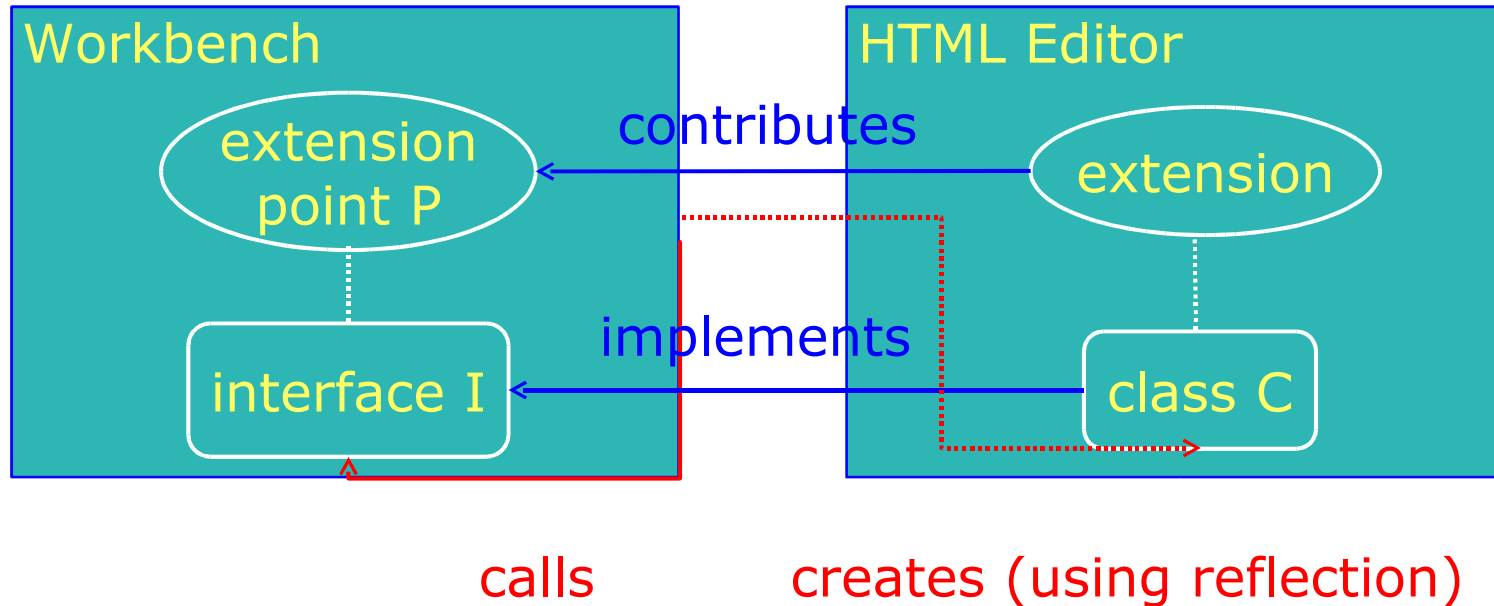
# The Demo

---

- File System Explorer providing a model of the file system, and views to browse the model
- No file viewers/editors are provided by the Explorer
- Several additional plug-ins to view/edit different kinds of files



# Contributing an Extension



- Workbench
  - Declares extension point P ([org.eclipse.ui.editors](http://org.eclipse.ui.editors))
  - Declares interface I ([IEditorPart](http://IEditorPart)) for P
- HTML-Editor
  - Implements interface I with its own class C ([HTMLEditor](http://HTMLEditor))
  - Contributes class C to extension point P
- Workbench instantiates HTMLEditor and calls its methods via interface IEditorPart



# Scalability

---

- The Workbench is highly scalable
- Leverages the extension point mechanism for progressive loading of code
- Supports dynamic plug-in addition
- Activities mechanism can be used for role-based “right-fitting” of UI
- Proven by successful products built on Eclipse:
  - WebSphere/Rational development tools
  - Lotus Workplace



# Optional RCP components

---

<b>Component</b>	<b>Description</b>
Help UI	Web-app-based Help UI
Update Manager	Discover and install new and updated versions of plugins
Text	Framework for high-function text editors
Forms	Forms-based control library
Welcome Page / Intro	Initial welcome experience and guided assistance
Cheat Sheets	Guides the user through a long-running, multi-step task such as a tutorial



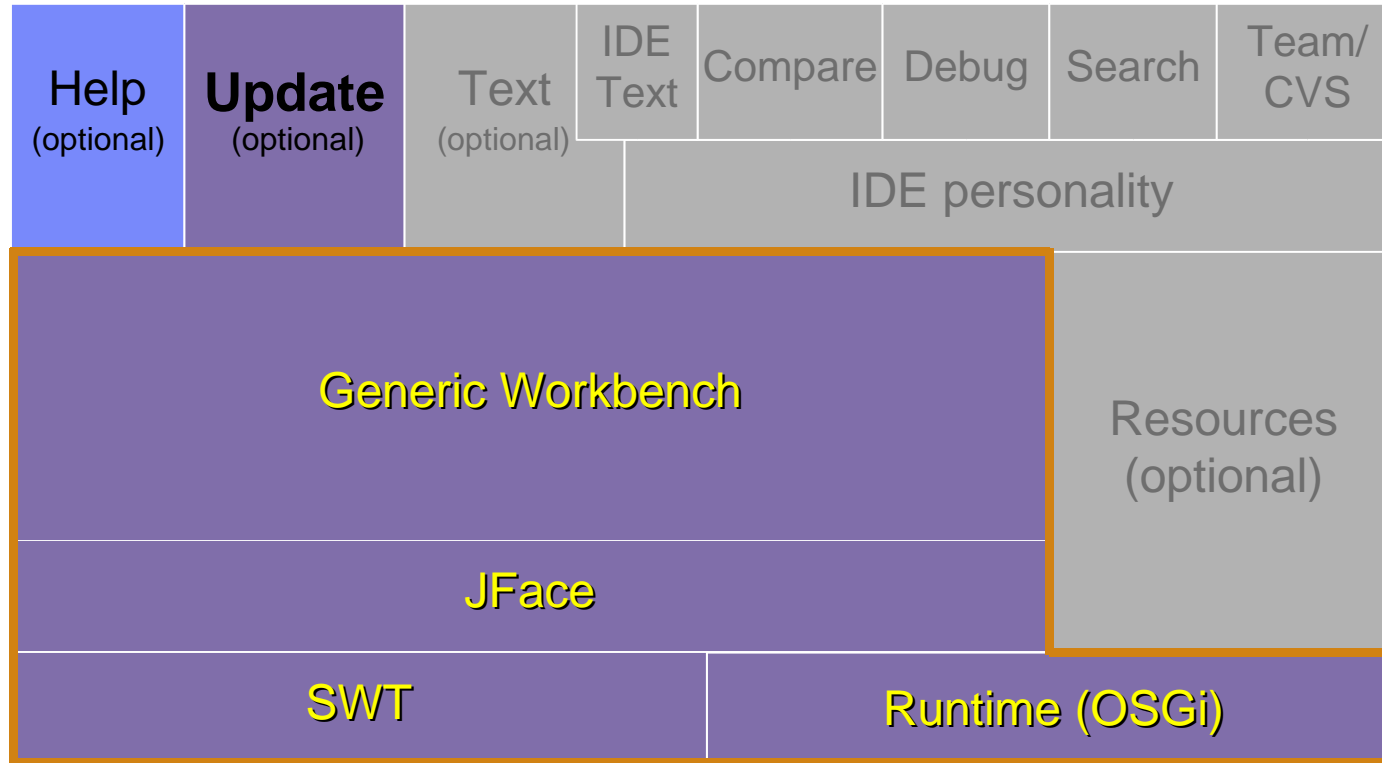
## Optional RCP components (cont'd)

---

<b>Component</b>	<b>Description</b>
Resources	Managed workspace with projects, folders, files, builders
Console, Outline, Properties views	Various extensible views
Graphical Editing Framework (GEF)	Includes Draw2D, a vector graphics framework
Eclipse Modeling Framework (EMF) and Service Data Objects (SDO)	EMF: Modeling framework and code generation facility based on a structured data model. SDO: Simplifies/unifies data application development in a service oriented architecture (SOA).



# Managing Plug-ins: Install/Update





## Managing Plug-ins: Install/Update

---

- **Features** group plug-ins into installable chunks
  - Feature manifest file
- Plug-ins and features bear version identifiers
  - major . minor . service
  - Multiple versions may co-exist on disk
- Features downloadable from URL addressable location
  - Using Eclipse Platform update manager
  - Obtain and install new plug-ins
  - Obtain and install patches & updates to existing plug-ins
- Support for update site mirroring & shared installations

# Patches

---

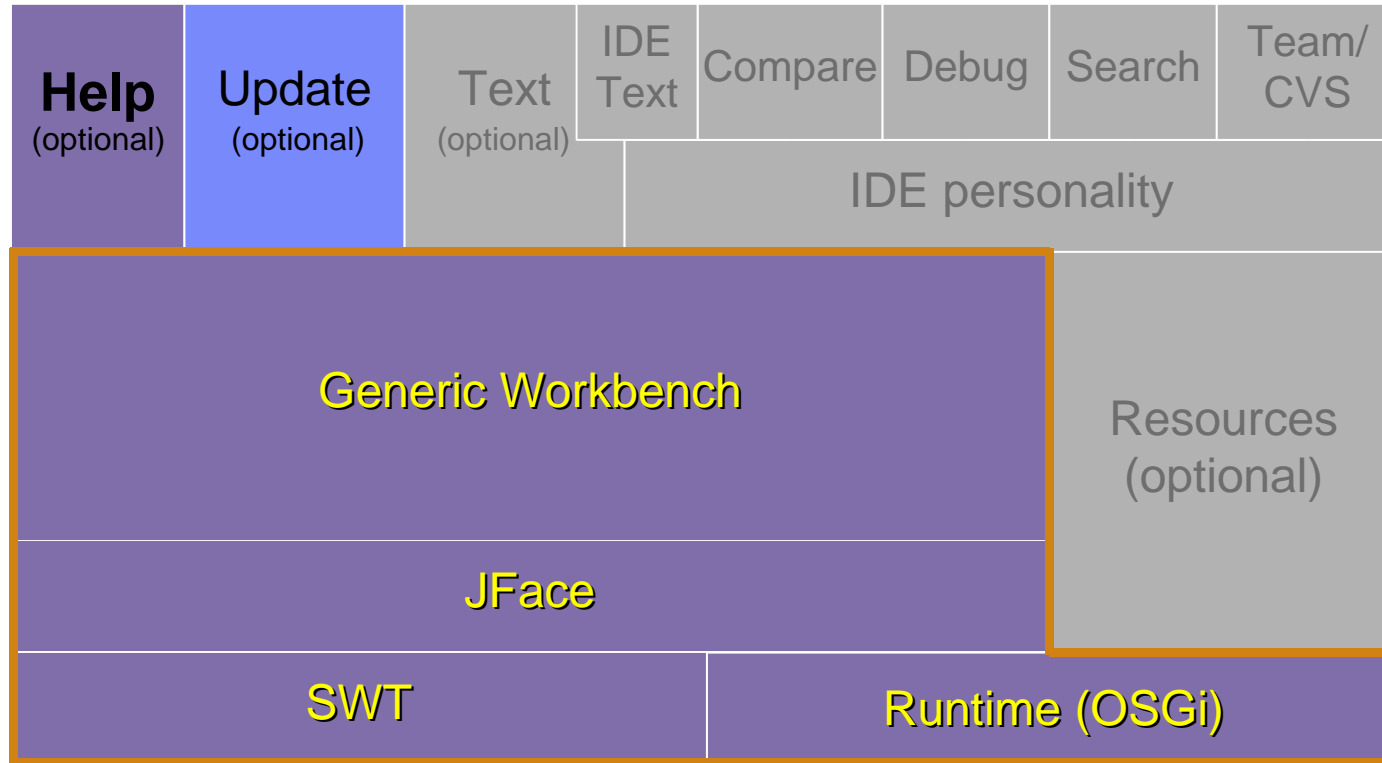
- Updates require that features are replaced with those of the same ID but a higher version
- For large products, it is hard to ship 'true' updates every time an emergency fix is needed
- Patches live side-by-side features they patch – they just bring new versions of select plug-ins (3.0 behavior)
- Patches contain either whole plug-ins or only those files that have changed
- Eclipse run-time sorts things out – picks the newer (patched) plug-ins





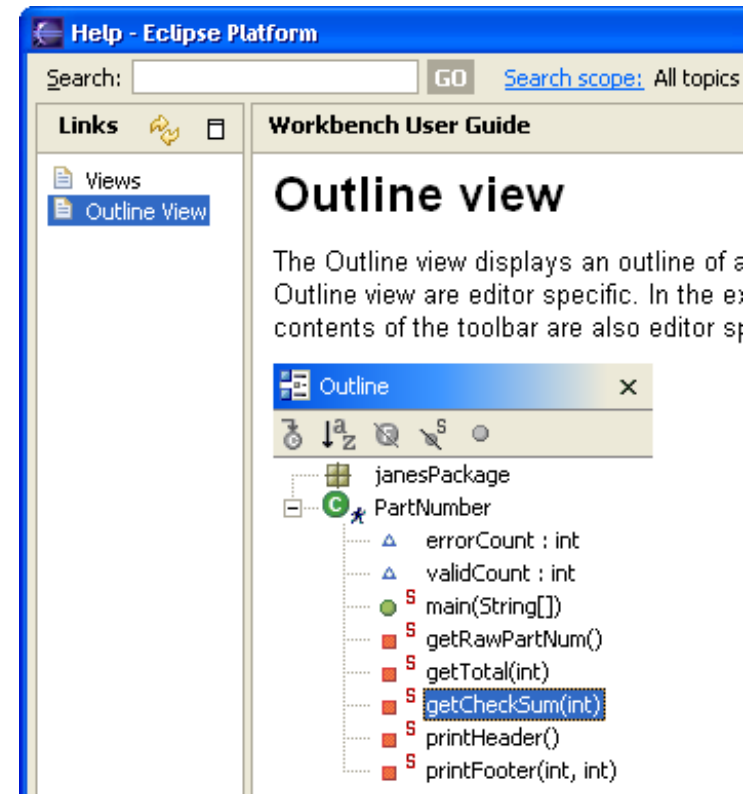
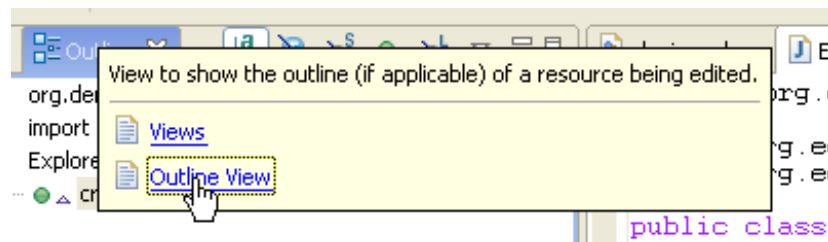
# Help System

---



# Help System

- Provides user help via F1 and Search
- Help files are written in HTML
- Content structure is defined in XML
- Help is presented in Web-Browser
- Highly scalable





## Developing Plug-ins: PDE

---

- PDE = Plug-in development environment
- Extenders use PDE to implement plug-ins
- Specialized tools for developing Eclipse plug-ins
- Built atop Eclipse Platform and JDT
  - Implemented as Eclipse plug-ins
  - Using Eclipse Platform and JDT APIs and extension points
- Features
  - Specialized PDE editor for plug-in manifest files
  - Templates for new plug-ins
  - PDE runs and debugs another Eclipse application



## Rich Client Platform Summary

---

- **Runtime:** Plug-in model and extension point architecture
- **OSGi:** Support for dynamic plug-ins
- **SWT:** Cross-platform native widget library, with tight OS integration
- **JFace:** UI framework to simplify common tasks
- **Workbench:** Highly scalable, managed UI
- Base RCP is relatively small: disk footprint is 5.5M
- Many optional components: Help UI, Update, Intro, Cheat Sheets, Forms, GEF, EMF, GEF, ...
- Tool support provided by PDE
- Solid architecture, proven by successful products
- Lots of documentation, and very good community support
- Opportunities for use of, and/or participation in, other Eclipse technology



## Where can I find out more ?

---

- RCP UI page:  
<http://www.eclipse.org/platform> > UI > RCP Home Page
- Ed Burnette's RCP tutorials
- Platform and RCP newsgroups:  
<news://news.eclipse.org/eclipse.platform.rcp>  
<news://news.eclipse.org/eclipse.platform>
- Gamma, Beck: Contributing to Eclipse – Principles, Patterns, and Plug-ins, Addison-Wesley, 2004  
[www.awprofessional.com/series/eclipse](http://www.awprofessional.com/series/eclipse)

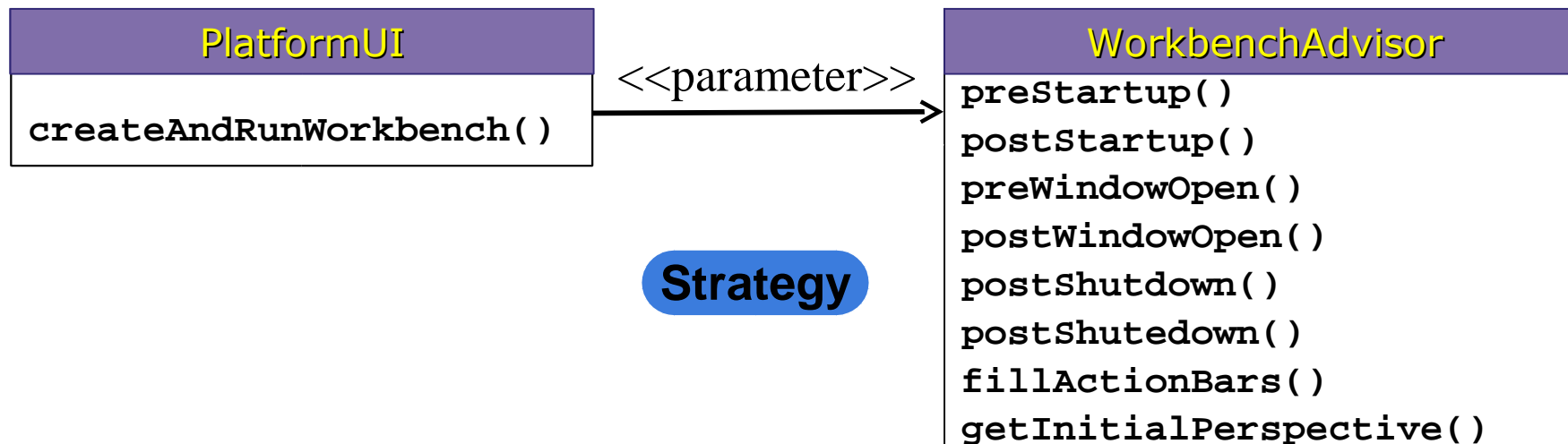


End of presentation. Code snippets follow.

---

# Configuring the Workbench Window

- WorkbenchAdvisor
  - a *strategy* object to configure a workbench window
  - provides hook methods called at strategic points during the workbench life cycle
  - defines the initial perspective





## A minimal WorkbenchWindow

---

```
class MinimalAdvisor extends WorkbenchAdvisor {  
  
    public void preWindowOpen(IWorkbenchWindowConfigurer configurer) {  
        super.preWindowOpen(configurer);  
        configurer.setShowCoolBar(true);  
    }  
  
    public void postWindowOpen(IWorkbenchWindowConfigurer configurer) {  
        super.postWindowOpen(configurer);  
        configurer.setTitle("File Explorer");  
    }  
  
    public String getInitialWindowPerspectiveId() {  
        return "org.demo.fileexplorer.workbench.explorerPerspective";  
    }  
}
```



# Defining the Perspective Layout

---

- contribute a perspective factory
  - hide the editor area

```
<extension point="org.eclipse.ui.perspectives">  
  <perspective name="File Explorer"  
    id="org.demo.fileexplorer.workbench.explorerPerspective"  
    class="org.demo.fileexplorer.workbench.ExplorerPerspective"/>  
</perspective>  
</extension>
```

```
class ExplorerPerspective implements IPerspectiveFactory {  
  public void createInitialLayout(IPageLayout layout) {  
    String editorArea = layout.getEditorArea();  
    layout.addView(ExplorerPlugin.DIRECTORY_VIEW,  
      IPageLayout.TOP, 0.33f, editorArea);  
    layout.addView(ExplorerPlugin.FILE_VIEW,  
      IPageLayout.RIGHT, 0.5f, ExplorerPlugin.DIRECTORY_VIEW);  
  }  
}
```

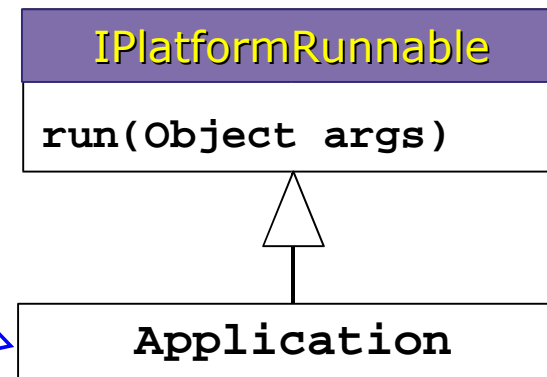
# Define an Eclipse Application

- We can run our contributions using the standard eclipse application
  - our contributions shows-up as part of the standard eclipse workbench
- define a custom entry point
  - *everything is a contribution*
 ...even the entry point into the eclipse platform

```

<extension id="application"
  point="org.eclipse.core.runtime.applications">
  <application>
    <run class="...workbench.Application"/>
  </application>
</extension>

```





# Launching the Application

---

- Run the workbench with the advisor

```
public class Application implements IPlatformRunnable {  
    public Object run(Object args) throws Exception {  
        WorkbenchAdvisor wa = new MinimalAdvisor();  
        Display d= PlatformUI.createDisplay();  
        int code= PlatformUI.createAndRunWorkbench(d, wa);  
        return new Integer(code);  
    }  
}
```

- Launch the application with your contributed application

```
java -cp startup.jar org.eclipse.core.launcher.Main  
    -application org.demo.fileexplorer.workbench.application
```