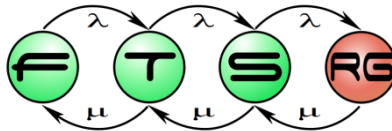


Introduction to Plug-in Development

Eclipse Based Technologies

<http://www.inf.mit.bme.hu/edu/courses/eat>



Eclipse: Open IDE and Platform

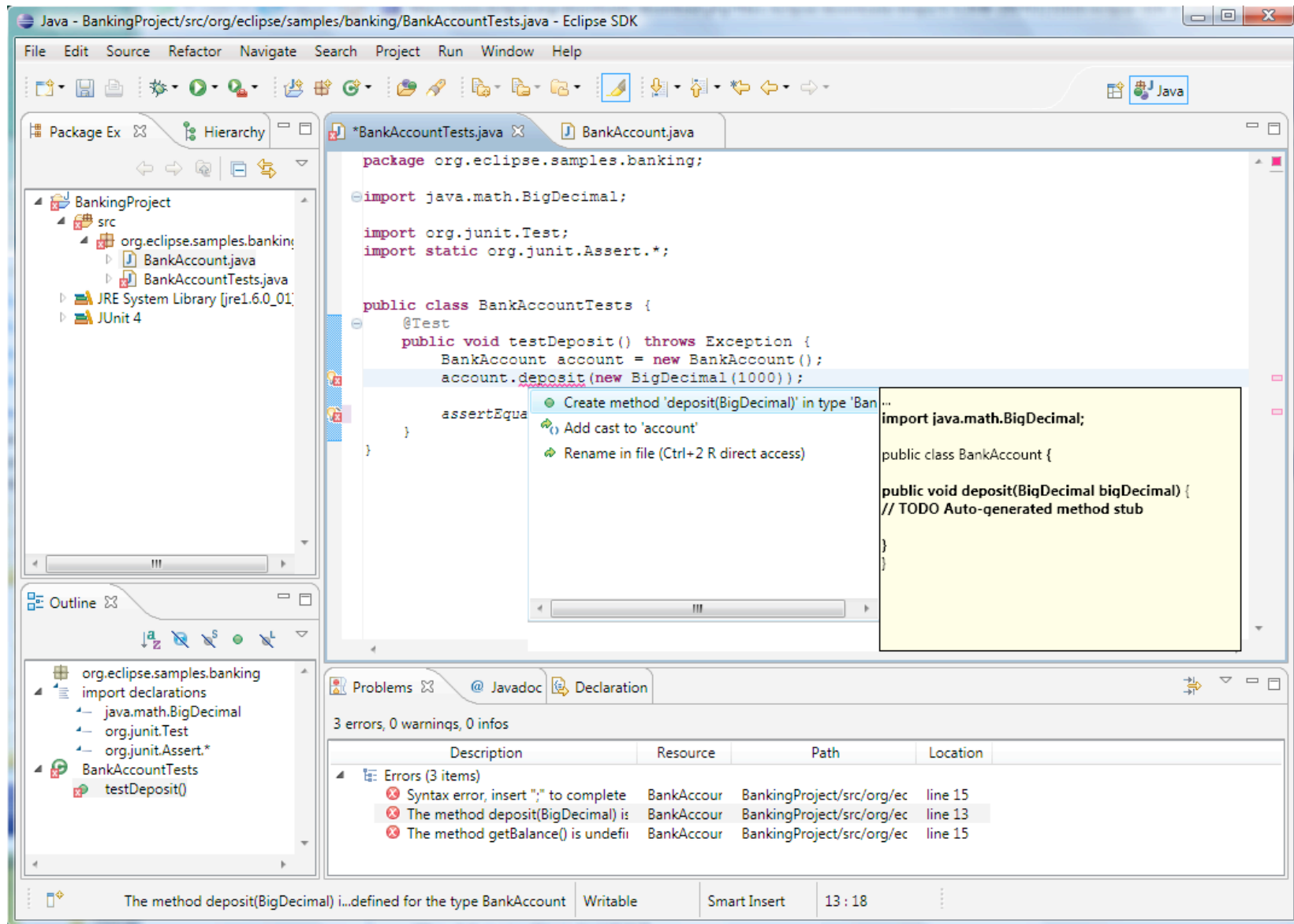
Why Eclipse?

- Strong industrial support
 - IBM, BEA, Oracle (partially)...
- Many successful development
 - Both industrial and research
- Well-designed architecture
 - Modular design (OSGi)
 - Application of design patterns

Why Eclipse?

- Integrated Development Environment (IDE)
 - Multiple languages (Java, C/C++, PHP, ...)
 - Graphical editors (pl. UML editor)
 - Connectors to issue trackers, ...
- Extensible application platform
 - Scales for very different environments
- Open source

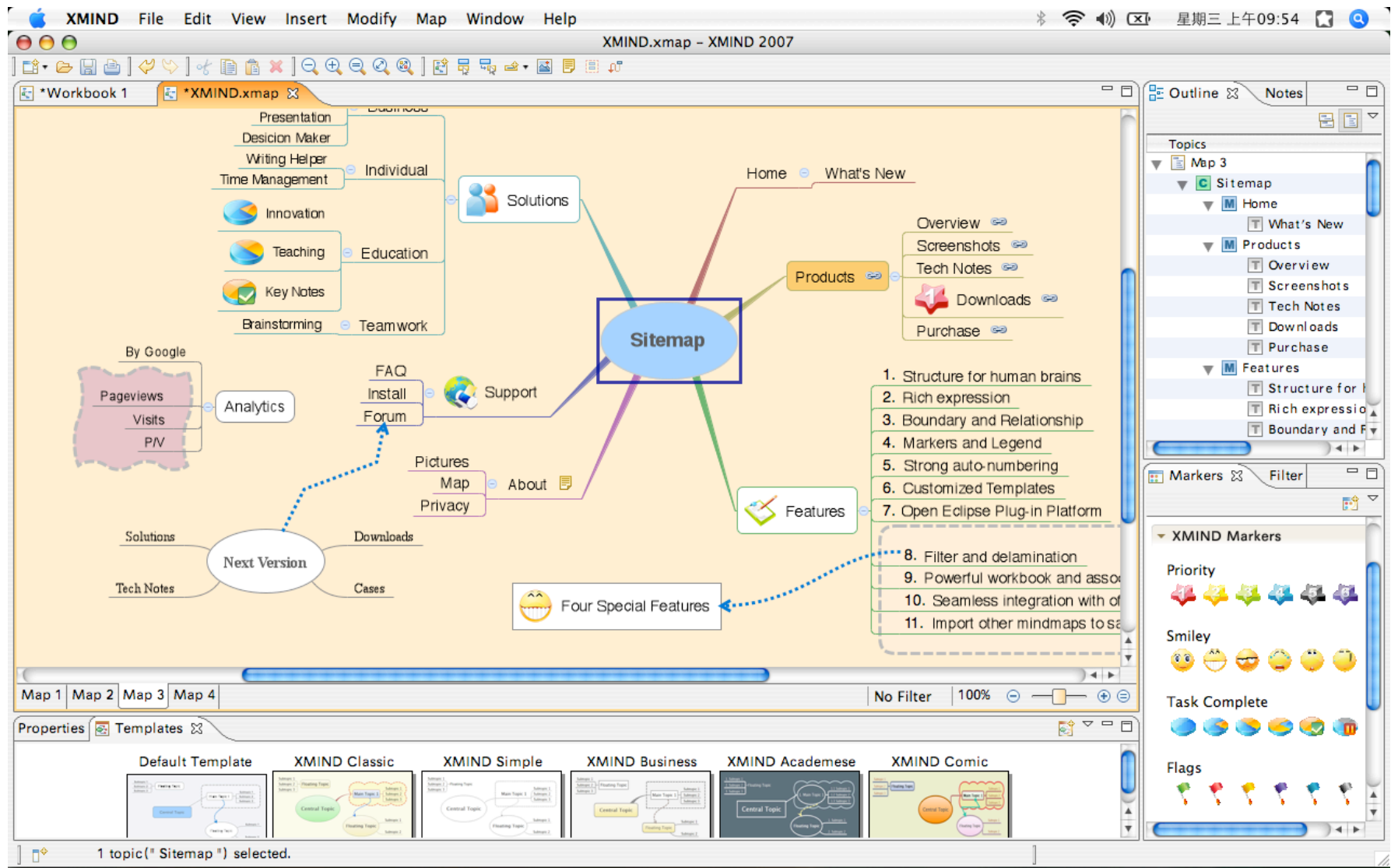
Eclipse: Development Environment



Eclipse: Application Platform

- Rich Client Platform (RCP)
 - Desktop application platform
 - Same extension mechanism as the IDE
 - IDE plug-ins reusable
 - In fact, Eclipse IDE is a specific RCP application
 - Framework for application integration
 - E.g., Lotus Symphony office software
- Rich Ajax Platform (RAP)
 - Web application platform
 - Goal: Single sourcing with RCP

RCP application - XMIND



RCP application- Kalypso

Hochwasser Vorhersage Team Synchronizing Kalypso Modeler

Navigator Style Editor

Style: Subcatchments

Regel: + + -

Subcatchments Subcatchments-Numb

Titel: Subcatchments

MinDenom: 0.0

MaxDenom: 77277.838217127

Symbolizer: Polygo Ort

Legende:

Polygon Line Point Text

Füllung

Fill-Farbe:

Fill-Opacity: 0.2

Outline

- ☐ + Legende
- ☒ + Knoten
- ☒ + VGewaesser
- ☒ + KMGewaesser - aktiv
- ☒ + RHBGewaesser
- ☒ + Teilgebiete
- ☐ + Subcatchments
- ☒ + Hydrotape
- ☐ + hydrotap

modell.gmv

NaModell0

- CatchmentCollectionMember
 - CatchmentCollection1
 - catchmentMember
 - Catchment100
 - Catchment101
 - Catchment102
 - bodenkorrekturmember
 - grundwasserabflussMember
 - entwaesserungsStrangMemb
 - KMChannel101
 - Catchment103
 - Catchment104
 - Catchment105
 - Catchment601
 - Catchment603
 - Catchment604
 - Catchment605
 - Catchment606
 - Catchment607
 - Catchment608

*Tabell... Tabelle... *Modell... *Karten... »1

*Tabelle_Teilgebiete.gtt

TG-Nummer (in...	Versiegelu...	Anstie...	TG-Flaeche (fla...	Faktor ...	Anfangsinhalt ...
100	0.188		1123080		
101	0.0080				
102					

<http://www.kalypso-simulation-platform.org>

RAP application– Yoxos OnDemand

Yoxos OnDemand – Get your personalized Eclipse

http://ondemand.yoxos.com ▶ geteclipse ▶ start

This web site does not supply identity information.

yoxos on demand
eclipse distribution
free eclipse download service

Components Templates

Search: Enter at least 3 characters
▶ Advanced

- Managed Templates
- C and C++ Development
- Charting and Reporting
- Communications
- Database Development
- Desktop
- Eclipse Development
- Graphical Editors and Frameworks
- Java Development
- Mobile Java
- Models and Model Development
- Mylyn
- Other Tools
- Programming Languages
- Quality Assurance
- Remote Access and Device Development
- Runtime
- SOA Development
- Science
- Source Code Management
- Sources
- Testing and Performance
- UI Development
- Web and Java EE Development
- Yoxos Tools

Plan

The following features will be added to your download for **MacOS X Cocoa**

No components are selected for installation.
Use the Add button to insert contents into the Plan.

Estimated download size: 0 kB

Add Shared Template Save As Template Start Download

Information License

Select elements from other views to display their information.

Get Certified Components.

Yoxos SecuSource
"Trusted Eclipse"

At a special introductory price until December 1st.

Feedback

Cheat Sheets

Yoxos OnDemand: Free Eclipse Download Service

Introduction

Discover the Yoxos OnDemand Services:
Create a custom download by selecting your favorite plugins or create a profile consisting of plugins AND team project sets, preferences, mylyn queries and more. We combine everything into a **single download** that you simply extract to create your personal Eclipse installation.

Click to Begin

- ▶ Selecting your Operating System
- ▶ Getting
- ▶ Automated dependency

The Plug-in Architecture of Eclipse

Extensible Application

Plug-in

Plug-in

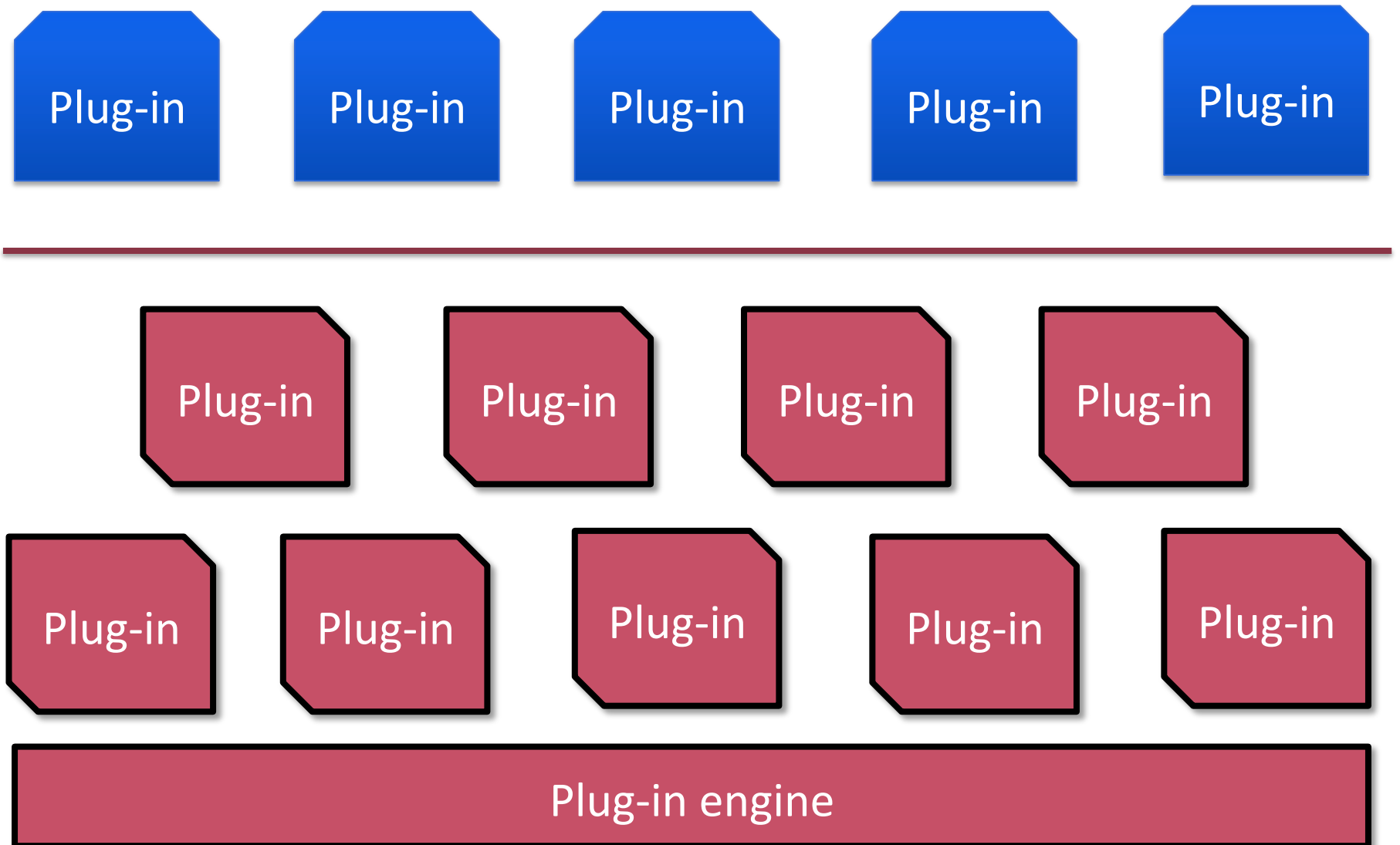
Plug-in

Plug-in

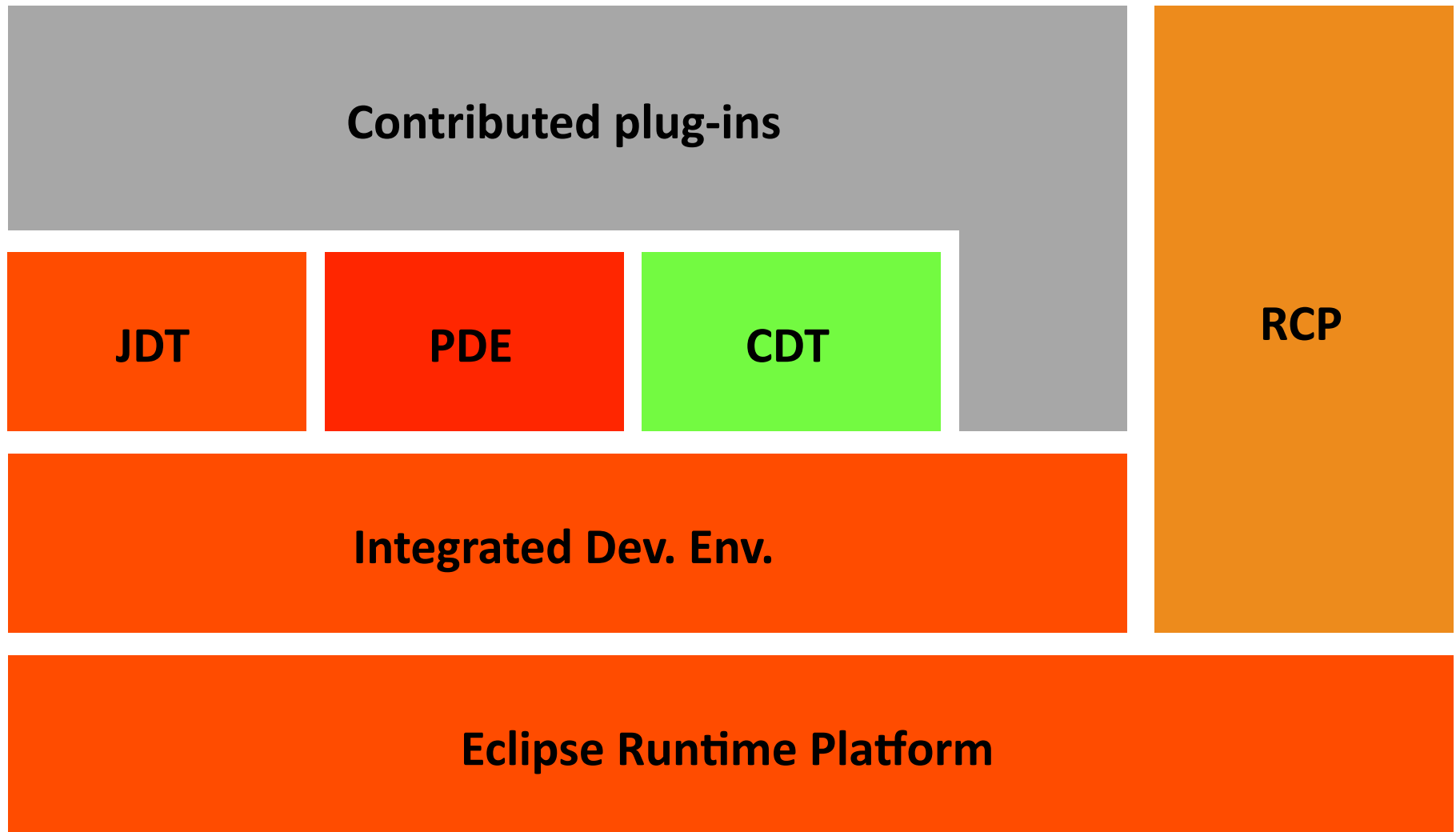
Plug-in

Application

Plug-in Based Application

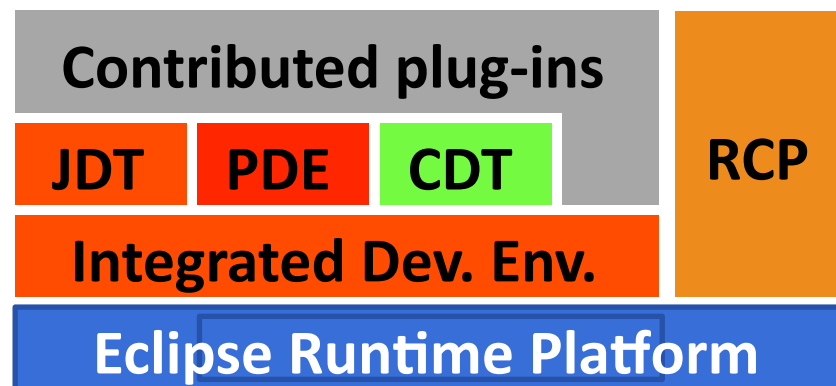


Eclipse Architecture



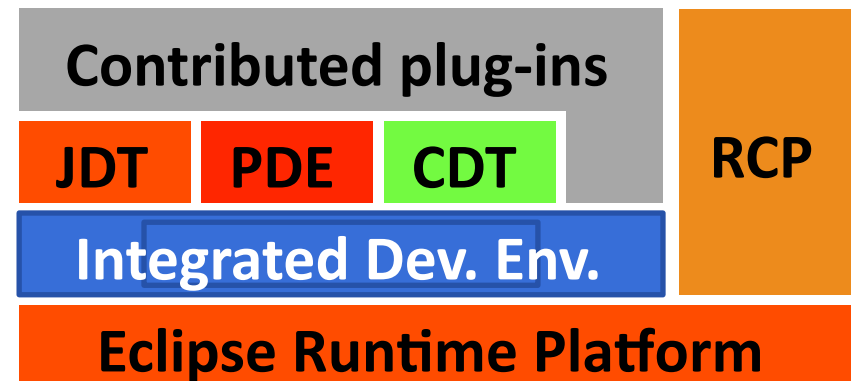
Eclipse Runtime Platform

- Plug-in engine
 - Platform startup
 - Plug-in registry
 - Resource management
- Software updates
- User interface
- Help engine



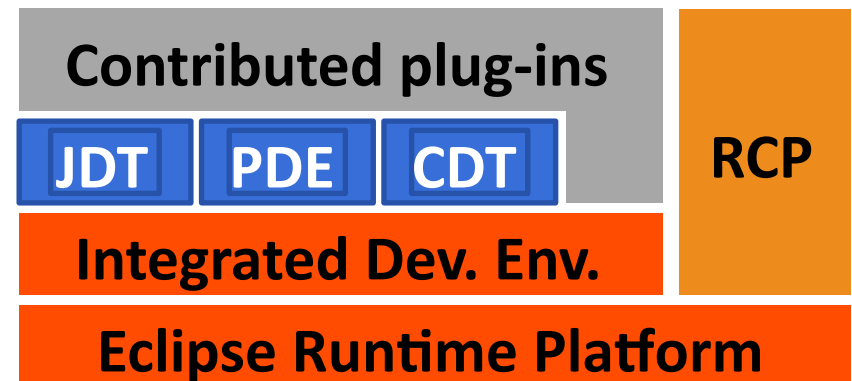
Language-Independent IDE

- Views, perspectives (see later)
- (Language-independent) Search Support
- (Language-independent) Debug Support
- Teamwork support



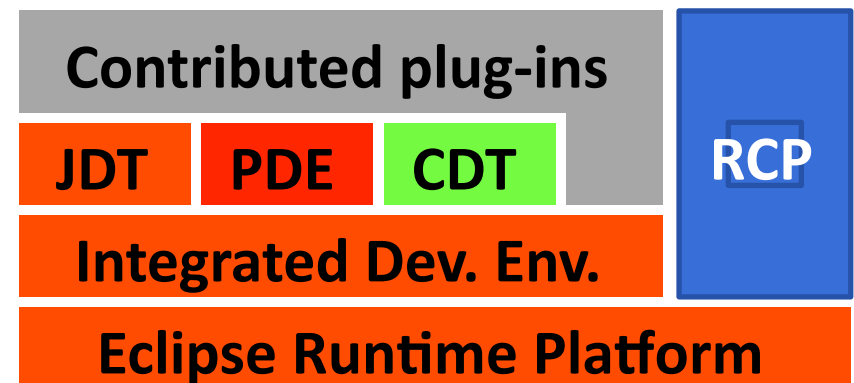
Language-specific Environments

- Java Development Toolkit (JDT)
 - Java Editor, Compiler, Debugger
 - Java Object Model/AST
- Plug-in Development Environment
 - Eclipse plug-in development
- C/C++ Development Tools
 - C/C++ editor
 - External debuggers
 - External compilers



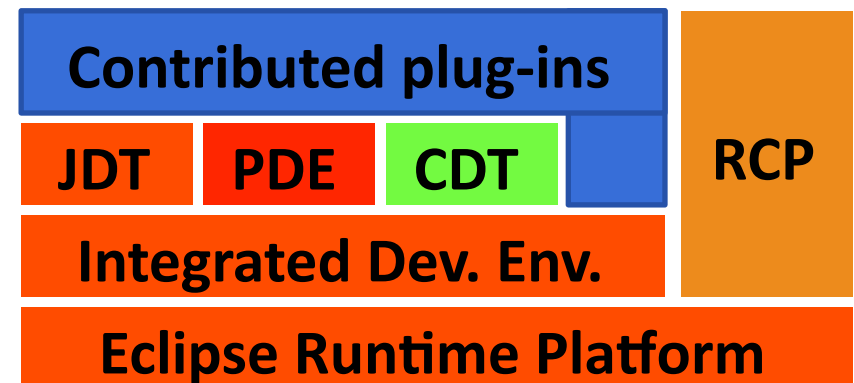
Eclipse Application Platform

- Application Platform
 - Re-useable Services
 - Platform 4.x introduces a new programming model (see later)
- Independent from IDE components
 - But they are re-usable if needed



Contributed Plug-ins

- Everything is possible (almost)
- Can re-use anything
 - Beware of dependencies!



Eclipse Plug-ins

Eclipse Plug-ins (Overview)

- Module concepts
 - Eclipse Plug-ins
 - *OSGi bundles (later in this course)*
- Co-operation
 - Calling API methods
 - Extension points
 - *OSGi services (later in this course)*

Eclipse Plug-in

- Smallest installable unit (sort of)
 - Java code
 - Metadata (descriptors)
- Packaging/distribution
 - jar archives
 - Recompiling existing libraries to plug-ins manageable
- Extension
 - Features: a group of plug-ins
 - Used at install (why?)

Lazy loading

- Basic rule
 - „Contributions are only loaded when they are needed.”
 - BUT: Plug-in metadata is always loaded

Most important metadata: Dependencies

Most important metadata: Dependencies

Plug-in 1

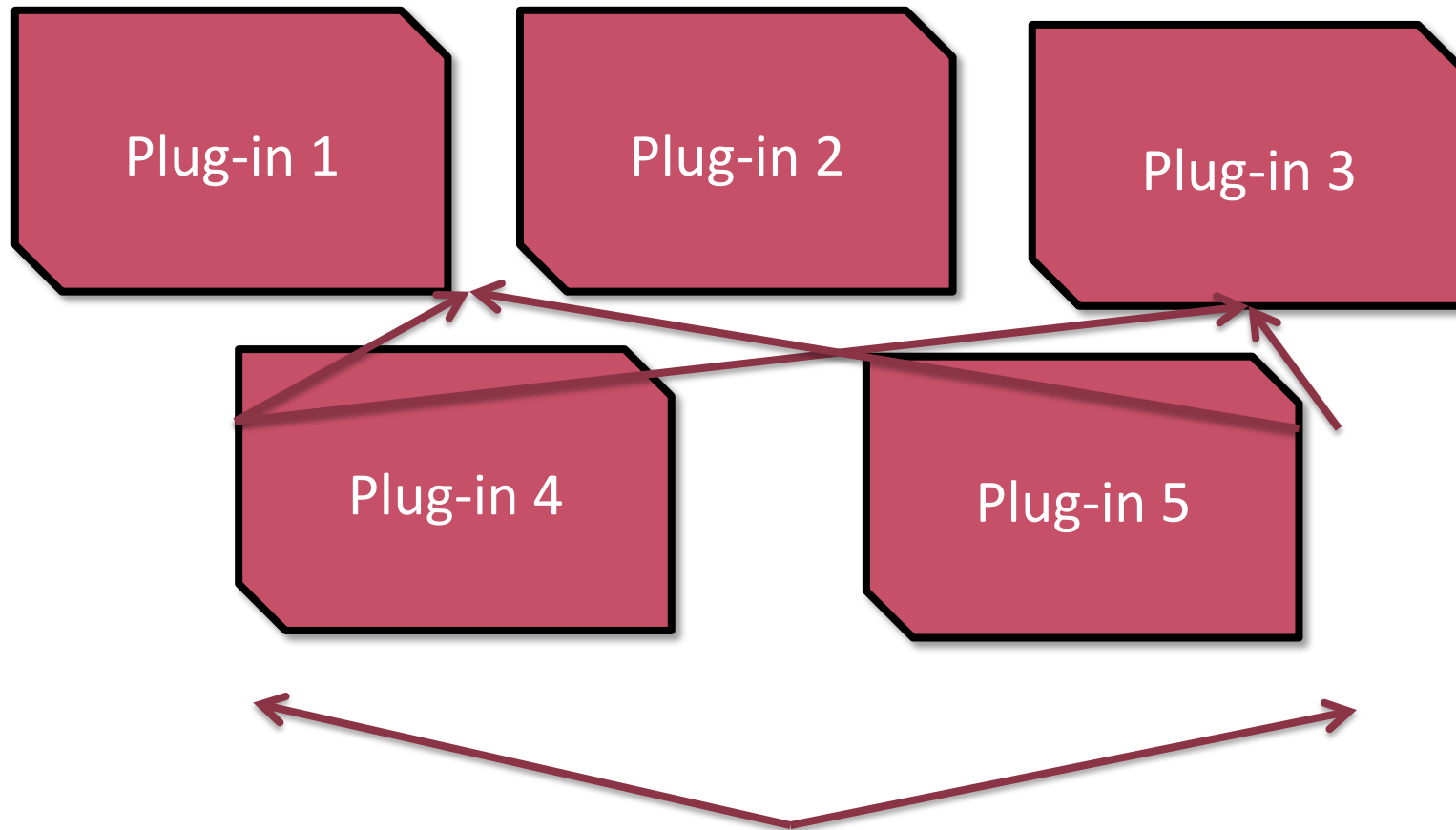
Plug-in 2

Plug-in 3

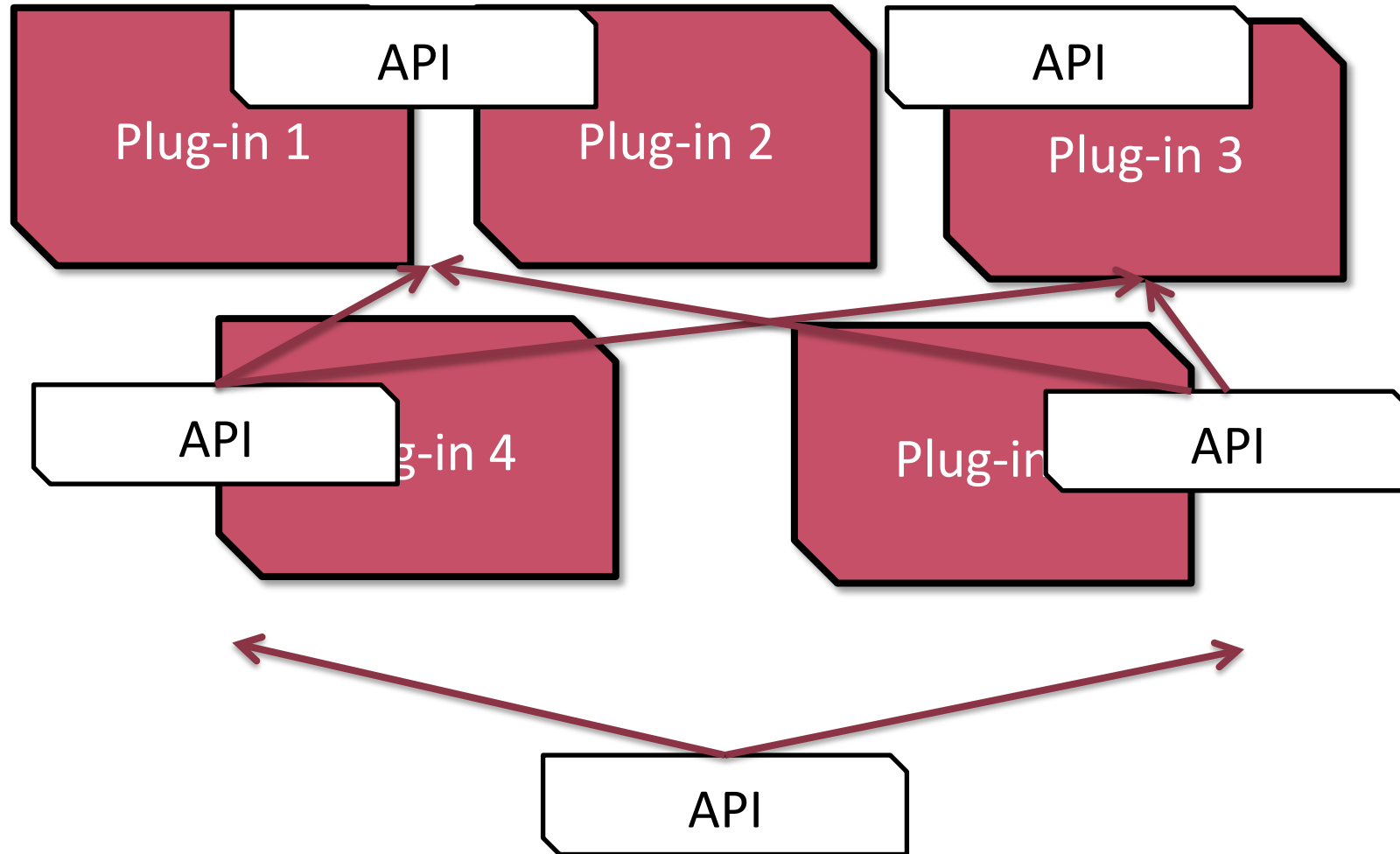
Plug-in 4

Plug-in 5

Most important metadata: Dependencies



Most important metadata: Dependencies



Standard Project Separation

- Core plug-in
 - Mandates API definition
- GUI plug-in
 - Allows headless execution if necessary
- Further plug-ins as necessary
 - In most cases, dependencies define boundaries
 - Avoid large plug-ins

API Definition

- List API packages (exported)
 - By default, packages are visible only inside the plug-in
 - Exported packages are also visible in other plug-ins
 - Via dependency definition
- Goal
 - Information hiding
 - Implementation classes not needed

API Design – Rules of Thumbs

- Explicit API: *Separate the API from internals*
 - Typically *.internal* packages
- Stability: *Once you invite others, don't change it*
 - API changes -> every user has to change
- Defensive API design:
 - *Reveal only the API in which you have confidence, but be prepared to reveal more API as clients ask for it.*

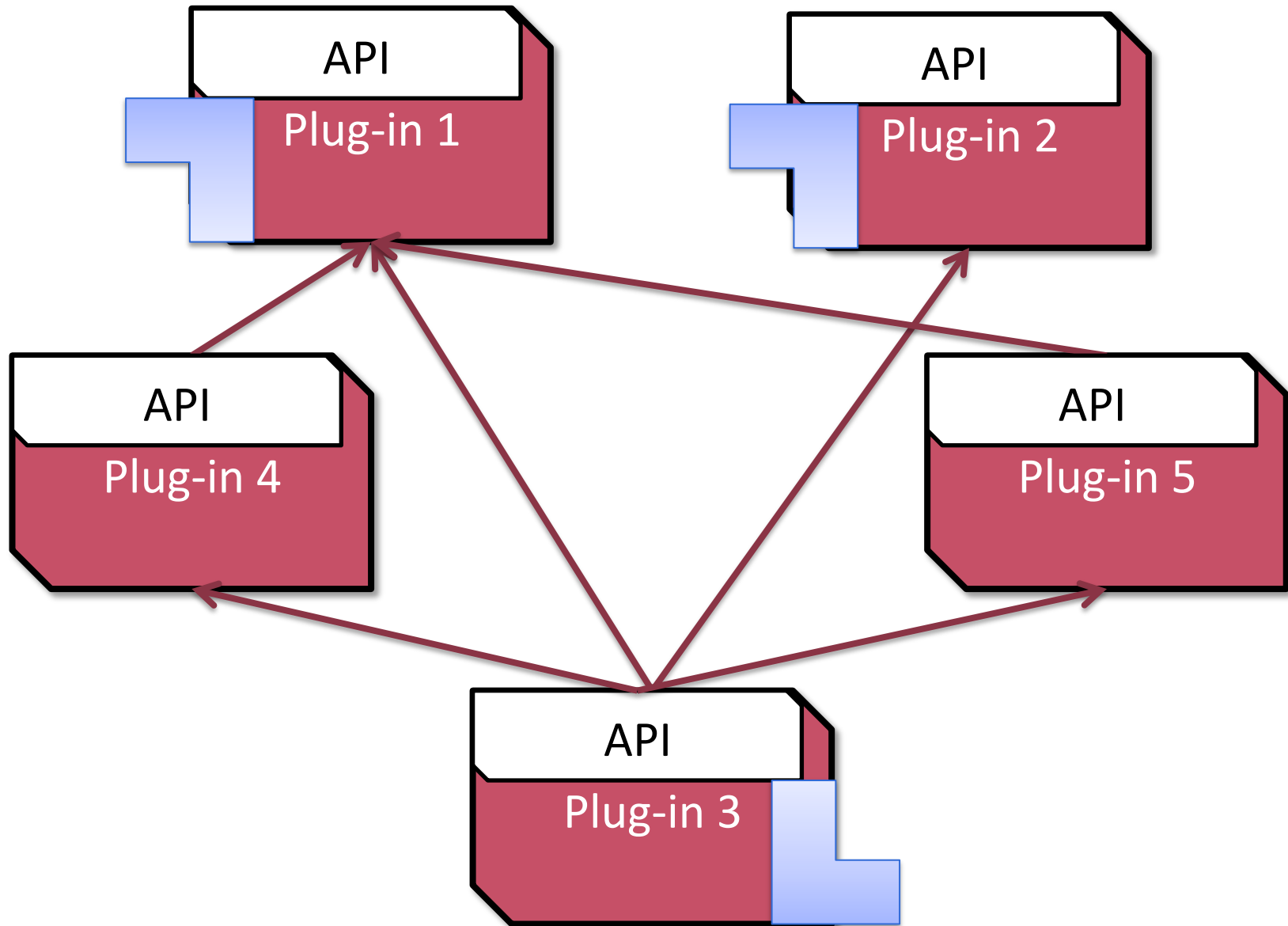
Problem with Generic Plug-ins

- Generic GUI extensions (e.g. menu items)
 - Unique labels and event handlers
 - **Must** be implemented in specific plug-ins
 - Unified presentation and management
 - **Should** be implemented in general plug-ins
- Problem
 - Generic plug-in **must not depend on** specific plug-ins
 - Dependency hierarchy/circles
- A solution
 - Extension points

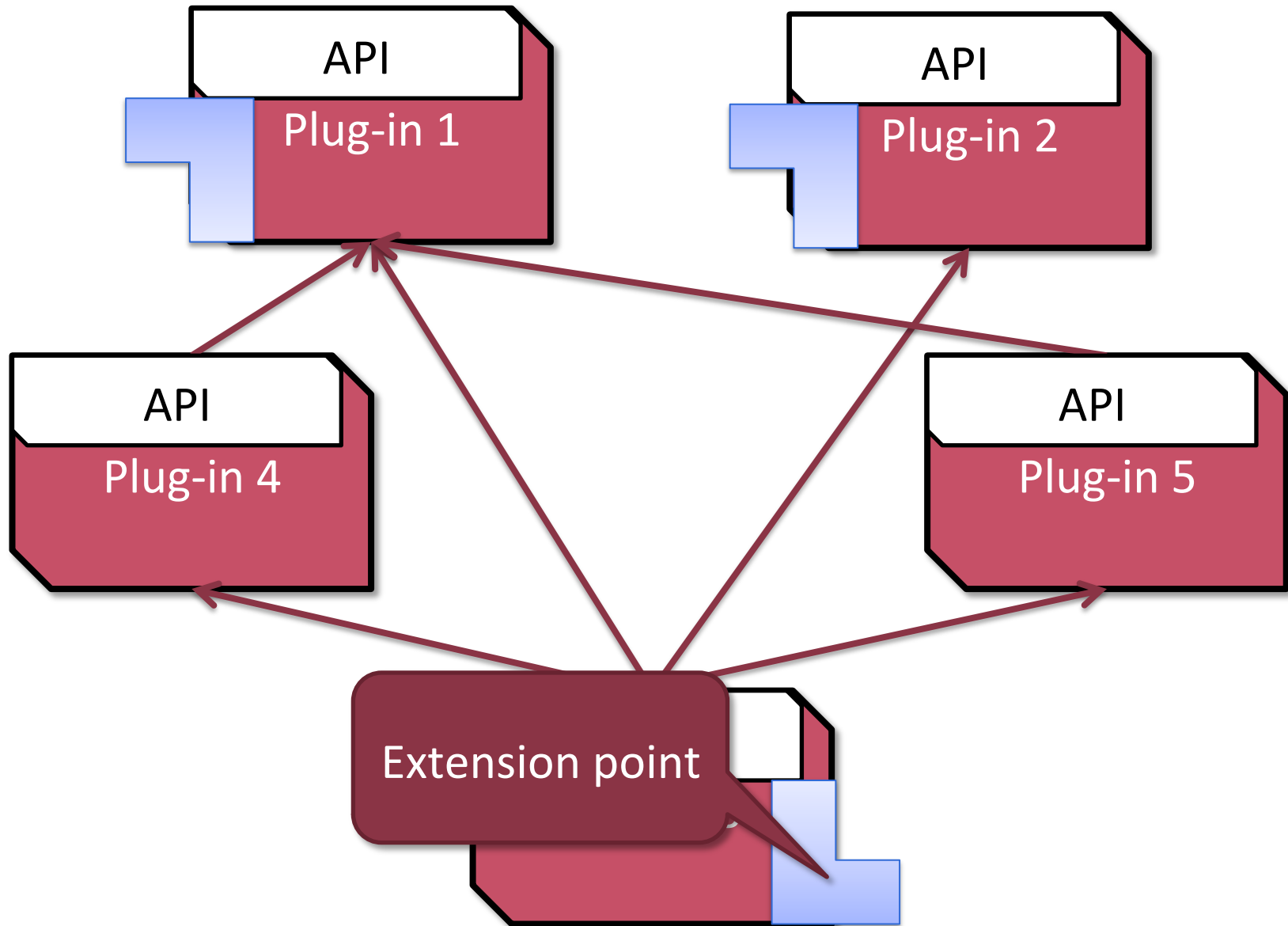
Extension Points

- Eclipse extension point
 - Definer: general plug-in
 - Extension point schema
 - Users: specific plug-ins
 - Configuration
 - Java code
 - Corresponds to schema
 - Dependency to the general plug-in
- Can instantiate non-API classes!

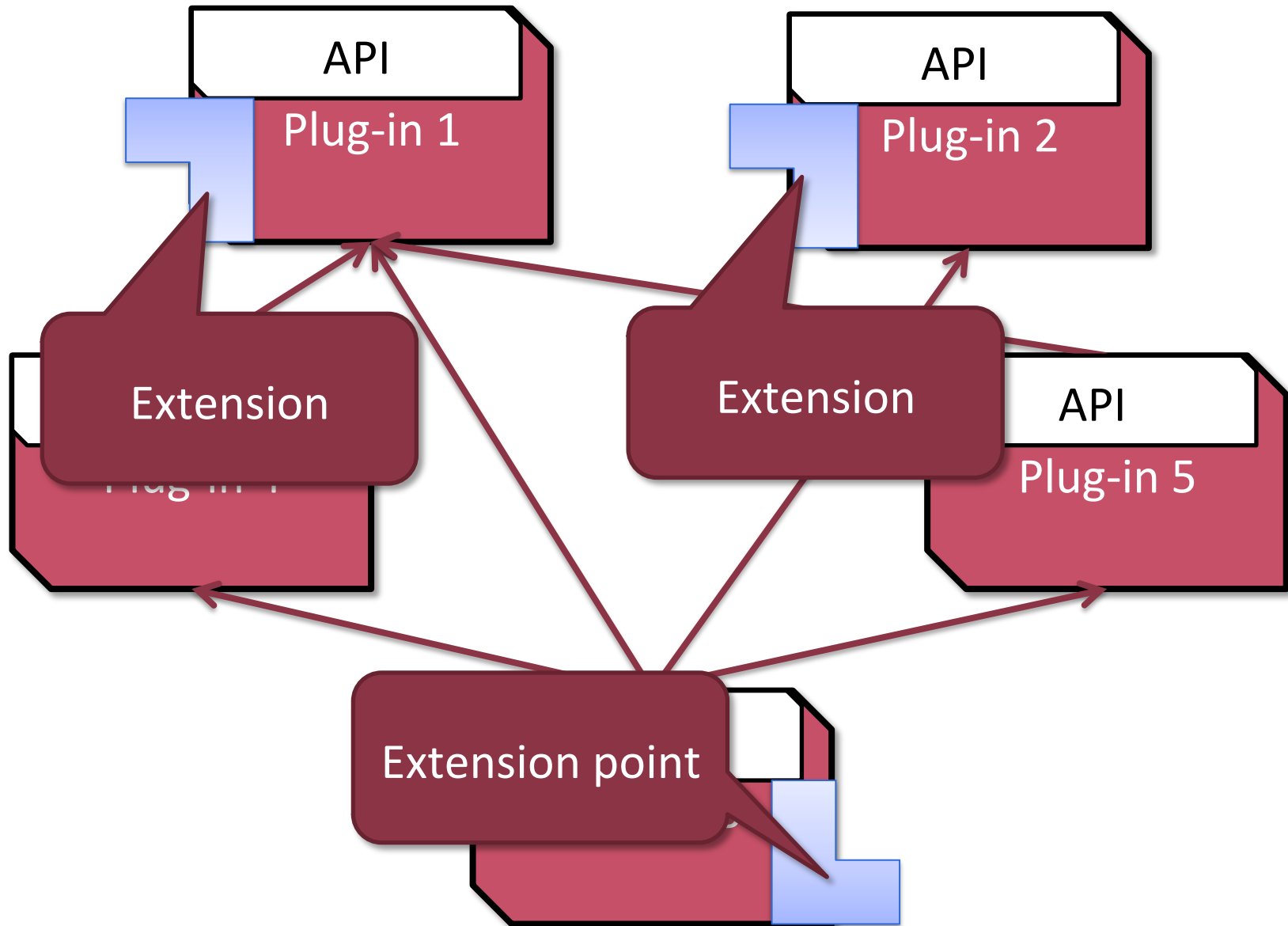
Extension point



Extension point



Extension point



Extension point – Rules of Thumb

- Error management
 - Do not assume correct extensions!
 - *As the provider of an extension point, you must protect yourself against misbehavior on the part of extenders*
- Invitation rule
 - *Whenever possible, let others contribute to your contributions.*
- Documentation
 - Always document the extension points

Extensions – Rules of Thumb

- *Contributions must conform to expected interfaces*
 - Extension must match the extension point definition
- *Share, don't replace*
 - An extension must work with other extensions
 - Do **not** try to exclude other extensions
 - Do not assume there are **no other extensions**
 - Exception:
 - Some extension point assume a single extension
 - Pl. org.eclipse.core.resources.teamHook
 - Extension point documentation!

Extension point - Implementation

- Using extensions

1. **IExtensionRegistry** instance:

`Platform.getExtensionRegistry()`

2. **IExtensionPoint** instance:

`registry.getExtensionPoint(String)`

3. **IExtension[]** instances:

`point.getExtensions()`

4. **IConfigurationElement** instance:

`extension.getConfigurationElements()`

ConfigurationElement

- Extension parameters
 - Tree structure (like the xml descriptor)
 - Can be validated (extension point schema)
- `String element.getAttribute(String)`
 - Attribute name as parameter
- `Object element.createExecutableExtension(String)`
 - Instantiating classes
 - Can instantiate non-exported classes as well
 - Only parameterless constructors can be used
 - Type casting required

Plug-in Rules of Thumb

- *“Monkey see/monkey do”*
 - *Always copy the structure of a similar plug-in*
 - Template wizards in Eclipse
- Relevance rule
 - *Contribute only when you can successfully operate*
 - Be careful to avoid cluttered user interface
- Follow platform conventions!
 - Re-use existing extensions/services
 - Copy the user interface (if possible)
 - User Interface Guidelines:
 - http://wiki.eclipse.org/User_Interface_Guidelines

Further Reading about Compatibility

- API Design and Evolution
 - <http://www.slideshare.net/moberhuber/eclipsecon-2010-api-design-and-evolution-tutorial>
- Evolving Java-based APIs
 - http://wiki.eclipse.org/index.php/Evolving_Java-based_APIS

Plug-in projects

Plug-in project structure

■ Metadata

- manifest.mf

- Name, version numbers and dependencies

- plugin.xml

- Optional
 - Extension and extension point definitions

■ Implementation

- Java code

■ Resources

- E.g., images, icons, configuration files

Plug-in project structure

■ Metadata

○ manifest.mf

- Name, version numbers and dependencies

○ plugin.xml

- Optional
- Extension and extension point definitions

■ Implementation

○ Java code

■ Resources

○ E.g., images, icons, configuration files

OSGi
convention

Plug-in project structure

■ Metadata

○ manifest.mf

- Name, version numbers and dependencies

OSGi
convention

○ plugin.xml

- Optional
- Extension and extension point definitions

Eclipse specific

■ Implementation

○ Java code

■ Resources

○ E.g., images, icons, configuration files

manifest.mf

- Plugin ID (mandatory)
 - Programmatic identification of the plug-in
- Plugin name (mandatory)
 - Name to display on user interface
- Version number (mandatory)
 - Semantic versioning (next slide)
- Dependencies
 - List of plug-ins (with version numbers)

Semantic versioning

- Fixed format: x.y.z.qualifier
 - x: **major** version
 - an increase signals compatibility breaking change
 - y: **minor** version
 - an increase signals backwards compatible changes
 - z: **patch** version
 - a bugfix release
 - **qualifier**
 - Build identifier number
 - Can be generated automatically
- Version comparison:
 - Comparing each version number as a string
- Example:
 - Last Eclipse 3.4 platform release:
 - 3.4.2.R200902111700

plugin.xml

- Extension point definition
 - XSD schema to define possible extensions
 - String literals, Java class or resource references
 - Mandatory/optional element definition
 - Documentation
- Extension definitions
 - Refers extension point
 - From any visible plug-in (including itself)
 - XML description validated by the extension point schema
 - Java class and resource references

plugin.xml example

```
<?eclipse version="3.4"?>
<plugin>
  <extension
    point="org.eclipse.ui.commands">
    <command
      description="Shows a Helloworld message."
      id="hu.bme.mit.commanddemo.helloworld"
      name="Display hello world!">
    </command>
  </extension>
  <extension
    point="org.eclipse.ui.handlers">
    <handler
      class="hu.bme.mit.commanddemo.commands.HelloworldHandler"
      commandId="hu.bme.mit.commanddemo.helloworld">
    </handler>
  </extension>
</plugin>
```


plugin.xml example

```
<?eclipse version="3.4"?>
```

```
<plugin>
```

```
  <extension
```

```
    point="org.eclipse.ui.commands">
```

```
    <command
```

```
      description="Shows a Helloworld message."
```

```
      id="hu.bme.mit.commanddemo.helloworld"
```

```
      name="Display hello world!">
```

```
    </command>
```

```
  </extension>
```

```
  <extension
```

```
    point="org.eclipse.ui.handlers">
```

```
    <handler
```

```
      class="hu.bme.mit.commanddemo.commands.HelloworldHandler"
```

```
      commandId="hu.bme.mit.commanddemo.helloworld">
```

```
    </handler>
```

```
  </extension>
```

```
</plugin>
```

Extension point definition

plugin.xml example

```
<?eclipse version="3.4"?>
<plugin>
  <extension
    point="org.eclipse.ui.commands">
    <command
      description="Shows a Helloworld message."
      id="hu.bme.mit.commanddemo.helloworld"
      name="Display hello world!">
    </command>
  </extension>
  <extension
    point="org.eclipse.ui.handlers">
    <handler
      class="hu.bme.mit.commanddemo.commands.HelloworldHandler"
      commandId="hu.bme.mit.commanddemo.helloworld">
    </handler>
  </extension>
</plugin>
```

Attribute

plugin.xml example

```
<?eclipse version="3.4"?>
<plugin>
  <extension
    point="org.eclipse.ui.commands">
    <command
      description="Shows a Helloworld message."
      id="hu.bme.mit.commanddemo.helloworld"
      name="Display hello world!">
    </command>
  </extension>
  <extension
    point="org.eclipse.ui.handlers">
    <handler
      class="hu.bme.mit.commanddemo.commands.HelloworldHandler"
      commandId="hu.bme.mit.commanddemo.helloworld">
    </handler>
  </extension>
</plugin>
```

Class reference

Form based metadata editor

The screenshot shows the 'Extensions' dialog for the 'hu.bme.mit.commanddemo' plug-in. The dialog is divided into two main sections: 'All Extensions' and 'Extension Details'.

All Extensions: This section allows defining extensions for the plug-in. It includes a text input field labeled 'type filter text' and a list of extension categories:

- ▶ org.eclipse.ui.commands
- ▶ org.eclipse.ui.menus
- ▶ org.eclipse.ui.handlers
- ▶ org.eclipse.ui.commandImages

Buttons for 'Add...', 'Remove', 'Up', and 'Down' are located to the right of the list.

Extension Details: This section is for setting the properties of the selected extension. It includes the following fields and instructions:

- Set the properties of the selected extension. Required fields are denoted by "*".**
- ID:**
- Name:**

Below the fields, there are three links with icons:

- [Show extension point description](#)
- [Open extension point schema](#)
- [Find declaring extension point](#)

The bottom of the dialog features a tabbed interface with the following tabs: Overview, Dependencies, Runtime, Extensions (selected), Extension Points, Build, MANIFEST.MF, plugin.xml, and build.properties.

Executing Eclipse Plug-ins

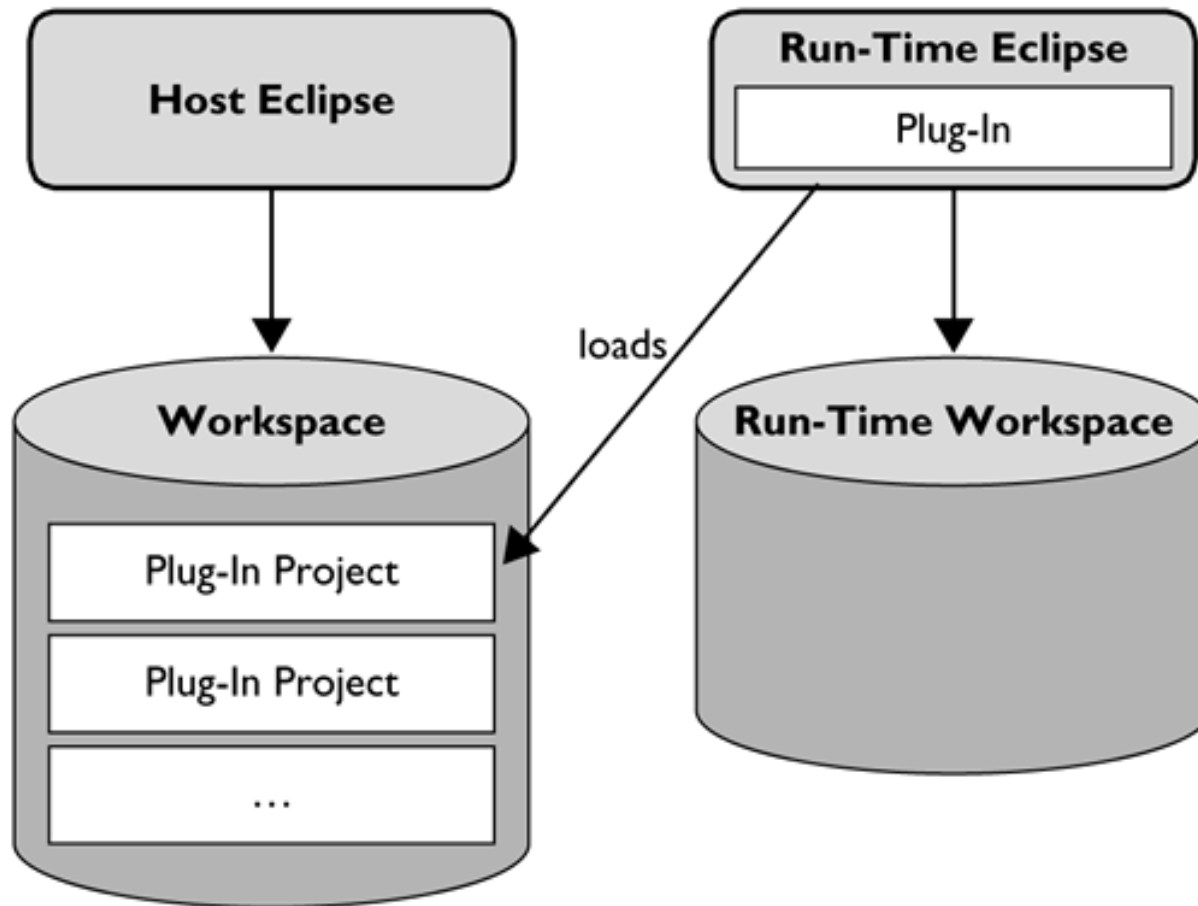
Target Platform

- A set of plug-ins
- Contains the developed plug-ins dependencies for
 - Compiling
 - Execution
- Different platforms for
 - IDE plug-ins
 - RCP applications
 - RAP applications
- Custom platforms possible

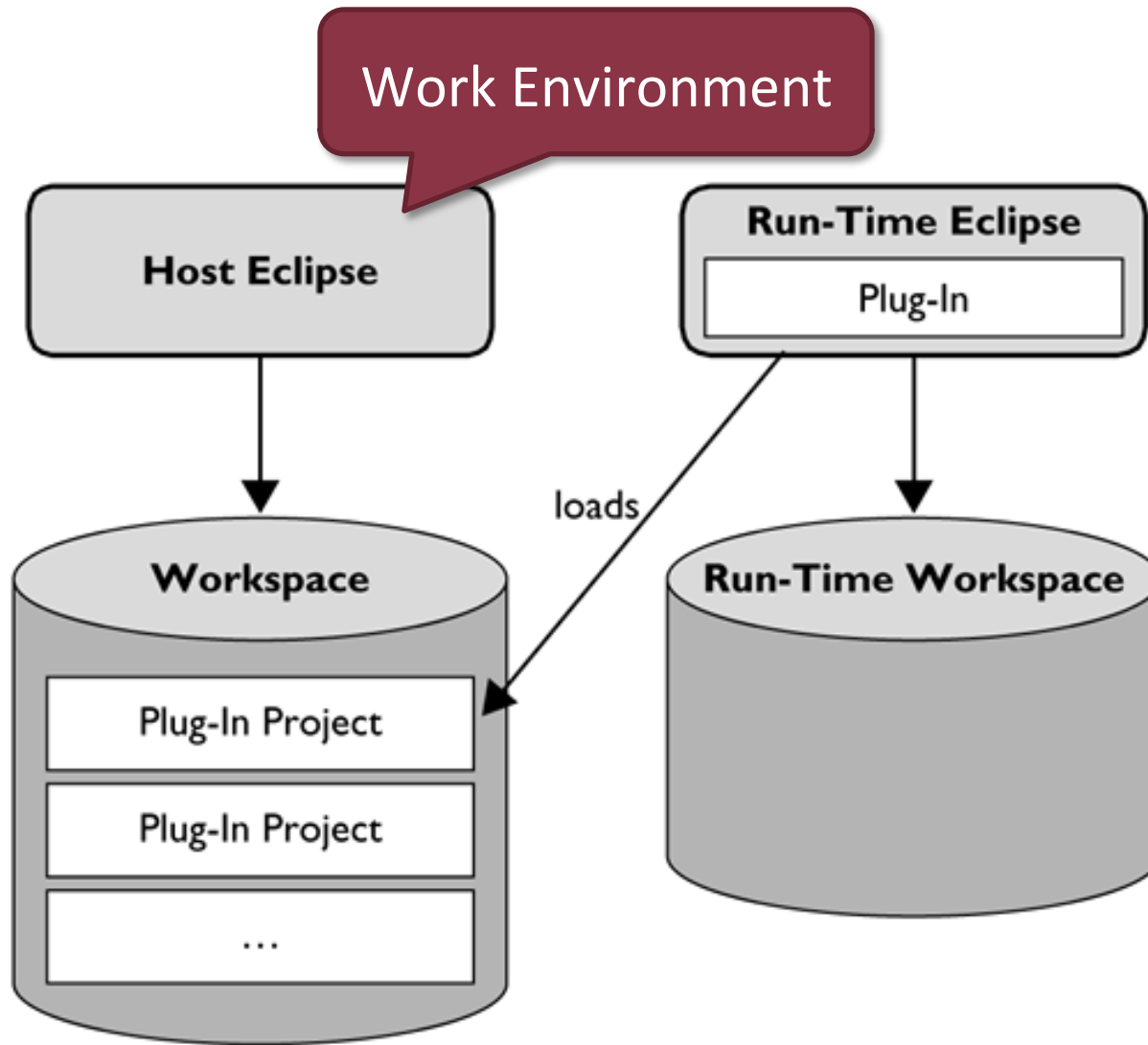
Plug-in Execution

- Execution classpath contains
 - Developed plug-ins
 - Target platform
- IDE plug-in
 - A new Eclipse workbench is started
- RCP/RAP application
 - The defined application starts

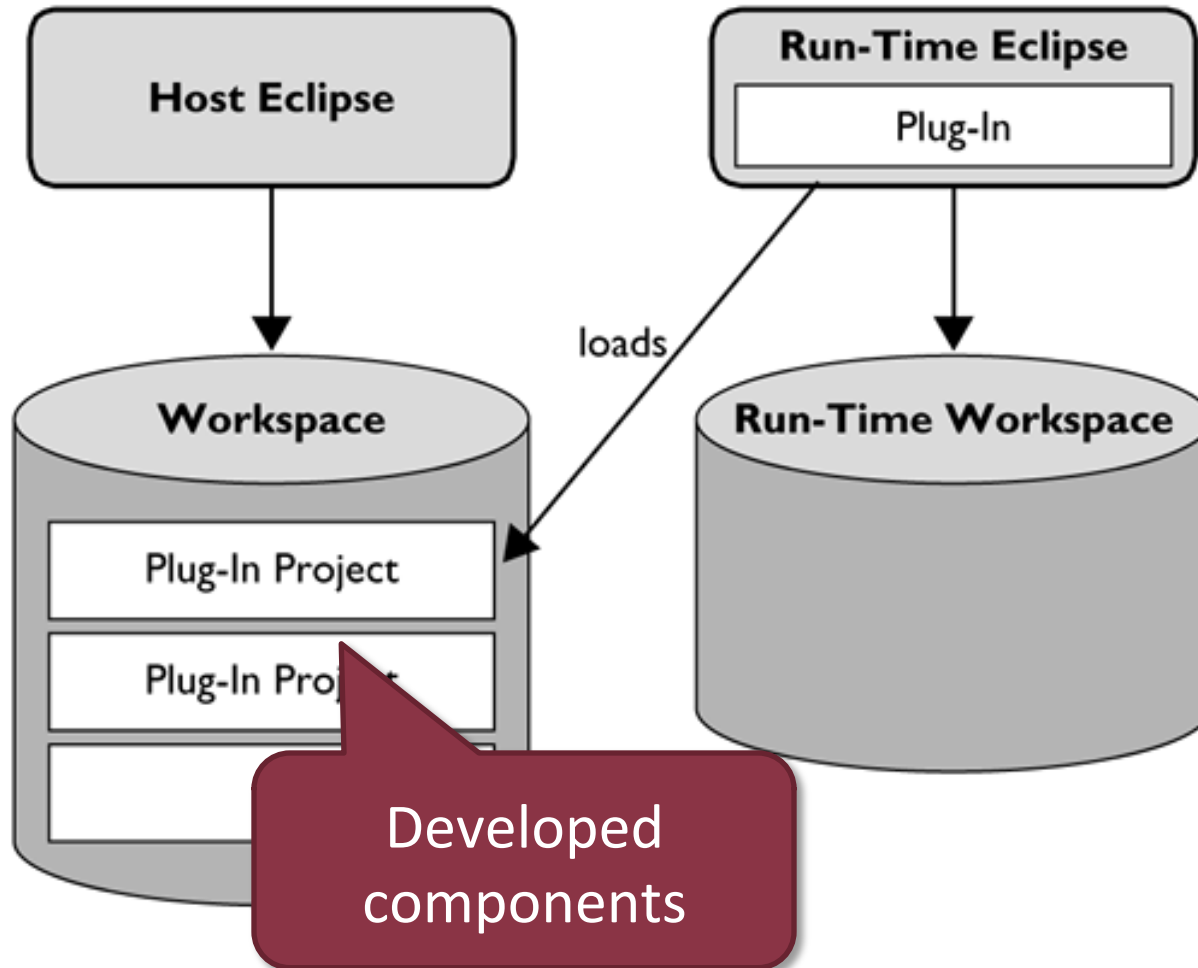
Execution



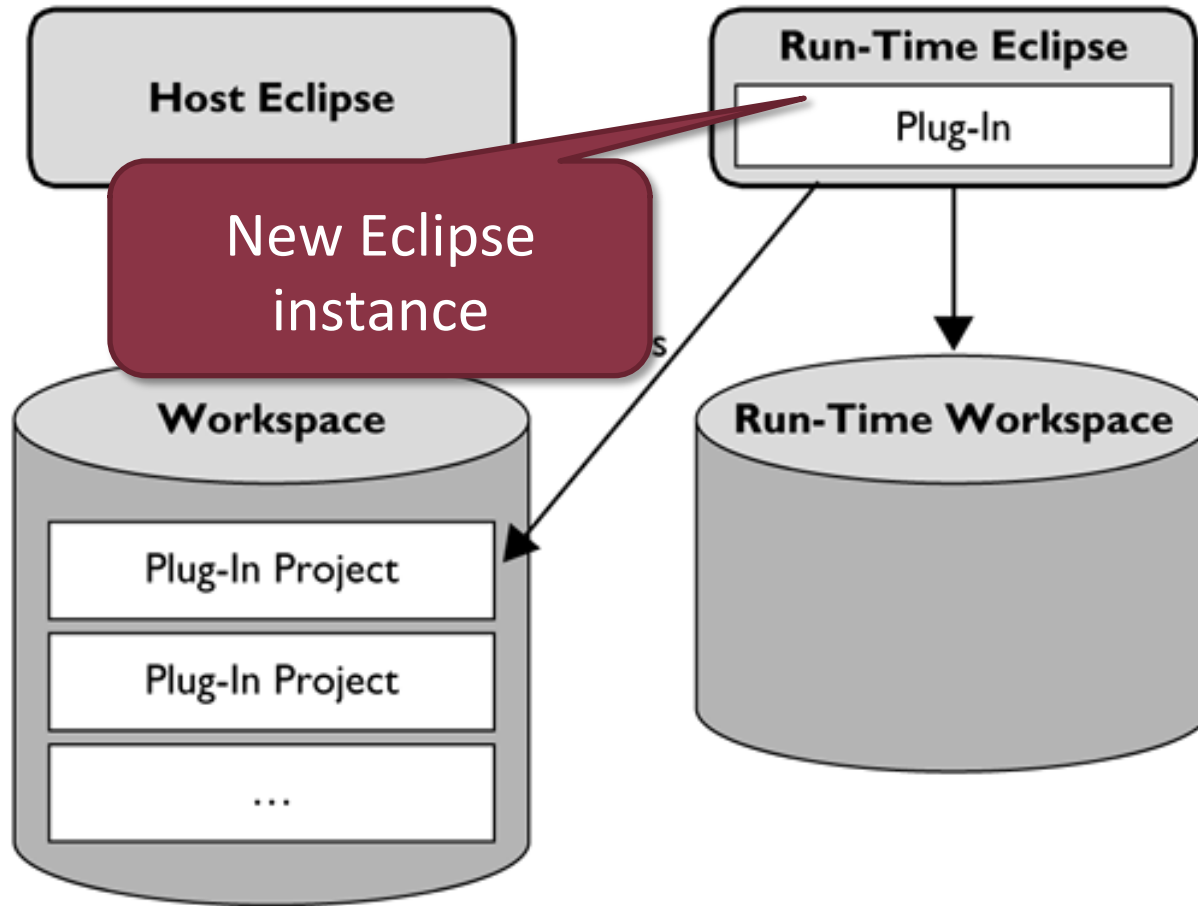
Execution



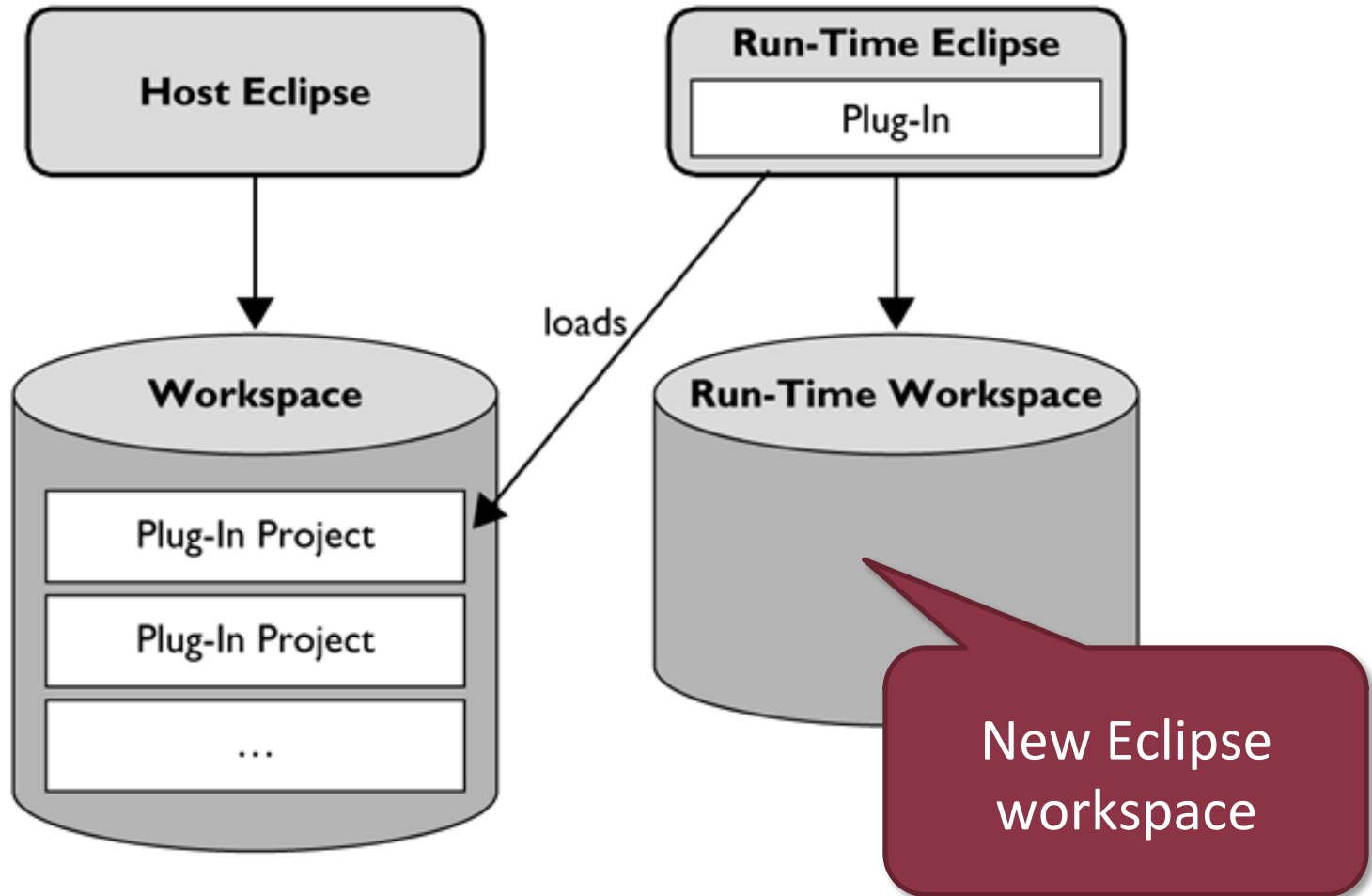
Execution



Execution



Execution



Common Extensions

Eclipse UI (3.x)

Workbench

- Workbench
 - A Window of an Eclipse-based application
 - Fixed set of elements supported
 - Configurable

Workbench structure

The screenshot displays the Eclipse IDE interface with the following components:

- Package Explorer:** Shows the project structure. The selected package is `hu.optxware.eclipsecourse.rcpdemo.gui`, which contains the following files:
 - `Activator.java`
 - `Application.java`
 - `ApplicationActionBarAdvisor.java`
 - `ApplicationWorkbenchAdvisor.java`
 - `ApplicationWorkbenchWindowAdvisor.java`
 - `BookListView.java`
 - `BookListViewContentProvider.java`
 - `Perspective.java`
 - `ServiceInterface.java`** (selected)
 - `ServiceManager.java`
- Main Editor:** Displays the code for `ServiceInterface.java`. The code is as follows:

```
package hu.optxware.eclipsecourse.rcpdemo.gui;

public interface ServiceInterface {

    public void serviceCreated();

    public void serviceRemoved();

}
```
- Outline:** Shows the `ServiceInterface` interface with its methods:
 - `serviceCreated() : void`
 - `serviceRemoved() : void`
- Problems:** Shows 0 items.

The status bar at the bottom indicates the editor is in **Writable** mode, with **Smart Insert** and a zoom level of **1 : 1**.

Workbench structure

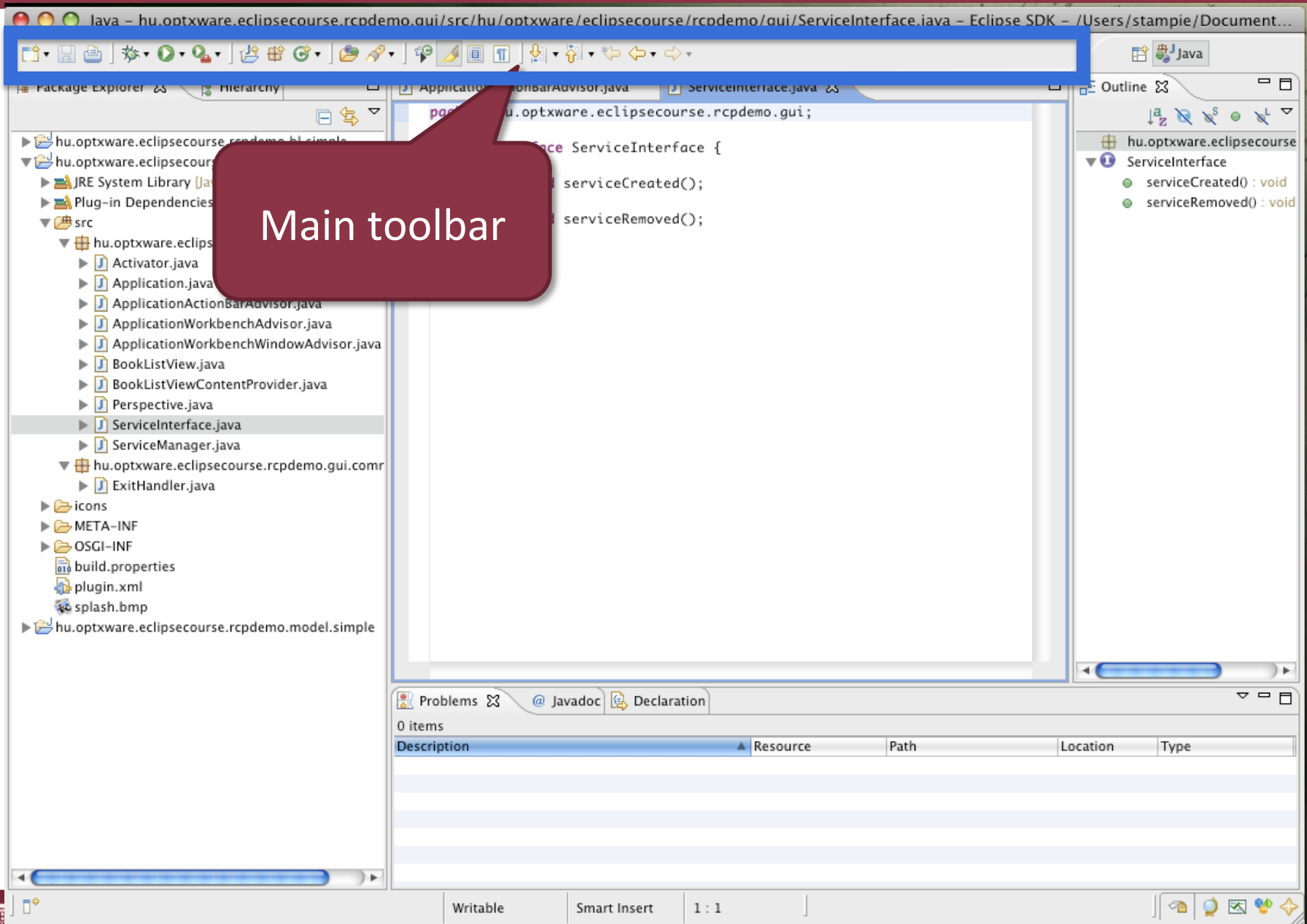
Workbench window

The screenshot displays the Eclipse IDE interface with the following components:

- Package Explorer (Left):** Shows the project structure for 'hu.optxware.eclipsecourse.rcpdemo.gui'. The 'src' folder is expanded, showing files like 'Activator.java', 'Application.java', 'ApplicationActionBarAdvisor.java', 'ApplicationWorkbenchAdvisor.java', 'ApplicationWorkbenchWindowAdvisor.java', 'BookListView.java', 'BookListViewContentProvider.java', 'Perspective.java', 'ServiceInterface.java' (selected), 'ServiceManager.java', and 'ExitHandler.java'. Other folders include 'icons', 'META-INF', 'OSGI-INF', 'build.properties', 'plugin.xml', 'splash.bmp', and 'hu.optxware.eclipsecourse.rcpdemo.model.simple'.
- Editor (Center):** Displays the code for 'ServiceInterface.java'. The code defines a 'ServiceInterface' with methods 'serviceCreated()' and 'serviceRemoved()'.
- Outline (Right):** Shows the 'ServiceInterface' class with its methods: 'serviceCreated() : void' and 'serviceRemoved() : void'.
- Bottom Panel:** Contains the 'Problems' view (showing 0 items), 'Javadoc' view, and 'Declaration' view. Below these is a table with columns: Description, Resource, Path, Location, and Type.

Description	Resource	Path	Location	Type

Workbench structure



Workbench structure

Workbench
Site

The screenshot displays the Eclipse IDE interface with the following components:

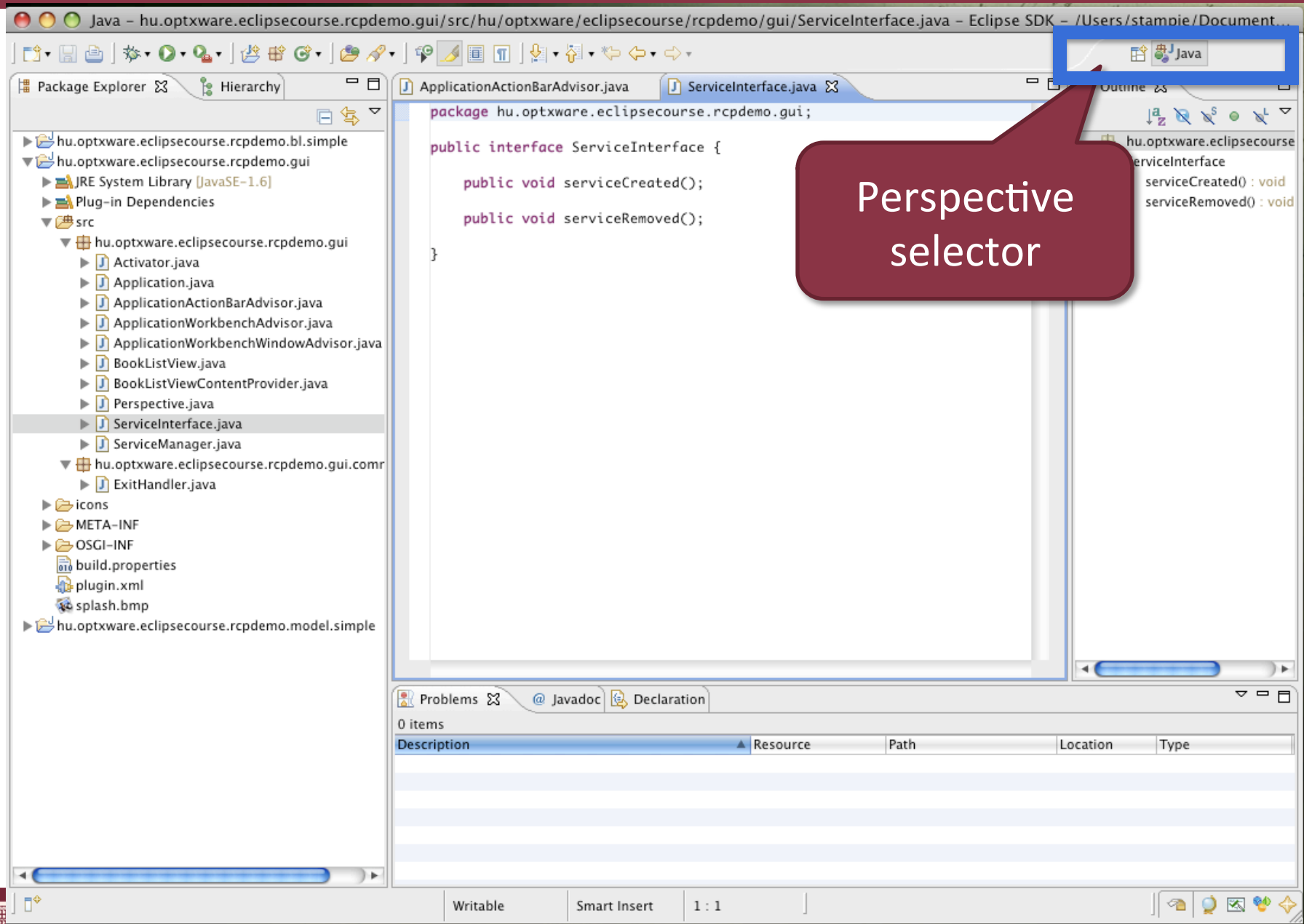
- Package Explorer (Left):** Shows a project hierarchy. The selected package is `hu.optxware.eclipsecourse.rcpdemo.gui`, which contains the file `ServiceInterface.java`.
- Editor (Center):** Displays the source code of `ServiceInterface.java`. The code defines a package `hu.optxware.eclipsecourse.rcpdemo.gui` and a public interface `ServiceInterface` with two methods: `serviceCreated()` and `serviceRemoved()`.
- Outline (Right):** Shows the structure of the `ServiceInterface` interface, listing the methods `serviceCreated() : void` and `serviceRemoved() : void`.
- Problems (Bottom):** Shows 0 items.
- Javadoc (Bottom):** Shows the Javadoc for the selected interface.
- Declaration (Bottom):** Shows the declaration of the selected interface.

Description	Resource	Path	Location	Type

Workbench Site

- Contains *Parts*
 - Editors (editor part)
 - Views (view part)
- Views are arranged according to selected *Perspective*

Workbench structure



Perspectives

- A functional group of Views
 - Layout description as well
 - Task-specific collection
- Examples
 - Java Development
 - Plug-in Development
 - Debug
 - Team Synchronizing

Perspective Implementation

- Perspective
 - Extension point: `org.eclipse.ui.perspectives`
 - Interface: `IPerspectiveFactory`
- Layouting support for views
 - Place all views related to editor area
- Further options
 - Perspective extensions can be used to extend an already defined view

Workbench structure

Java – hu.optxware.eclipsecourse.rcpdemo.gui/src/hu/optxware/eclipsecourse/rcpdemo/gui/ServiceInterface.java – Eclipse SDK – /Users/stampie/Document...

Package Explorer Hierarchy

- hu.optxware.eclipsecourse.rcpdemo.bl.simple
- hu.optxware.eclipsecourse.rcpdemo.gui
 - JRE System Library [JavaSE-1.6]
 - Plug-in Dependencies
 - src
 - hu.optxware.eclipsecourse.rcpdemo.gui
 - Activator.java
 - Application.java
 - ApplicationActionBarAdvisor.java
 - ApplicationWorkbenchAdvisor.java
 - ApplicationWorkbenchWindowAdvisor.java
 - BookListView.java
 - BookListViewContentProvider.java
 - Perspective.java
 - ServiceInterface.java
 - ServiceManager.java
 - hu.optxware.eclipsecourse.rcpdemo.gui.com
 - ExitHandler.java
 - icons
 - META-INF
 - OSGI-INF
 - build.properties
 - plugin.xml
 - splash.bmp
- hu.optxware.eclipsecourse.rcpdemo.model.simple

ApplicationActionBarAdvisor.java ServiceInterface.java

```
package hu.optxware.eclipsecourse.rcpdemo.gui;

public interface ServiceInterface {

    public void serviceCreated();

    public void serviceRemoved();

}
```

Outline

- hu.optxware.eclipsecourse
 - ServiceInterface
 - serviceCreated() : void
 - serviceRemoved() : void

Views (0..N) in groups

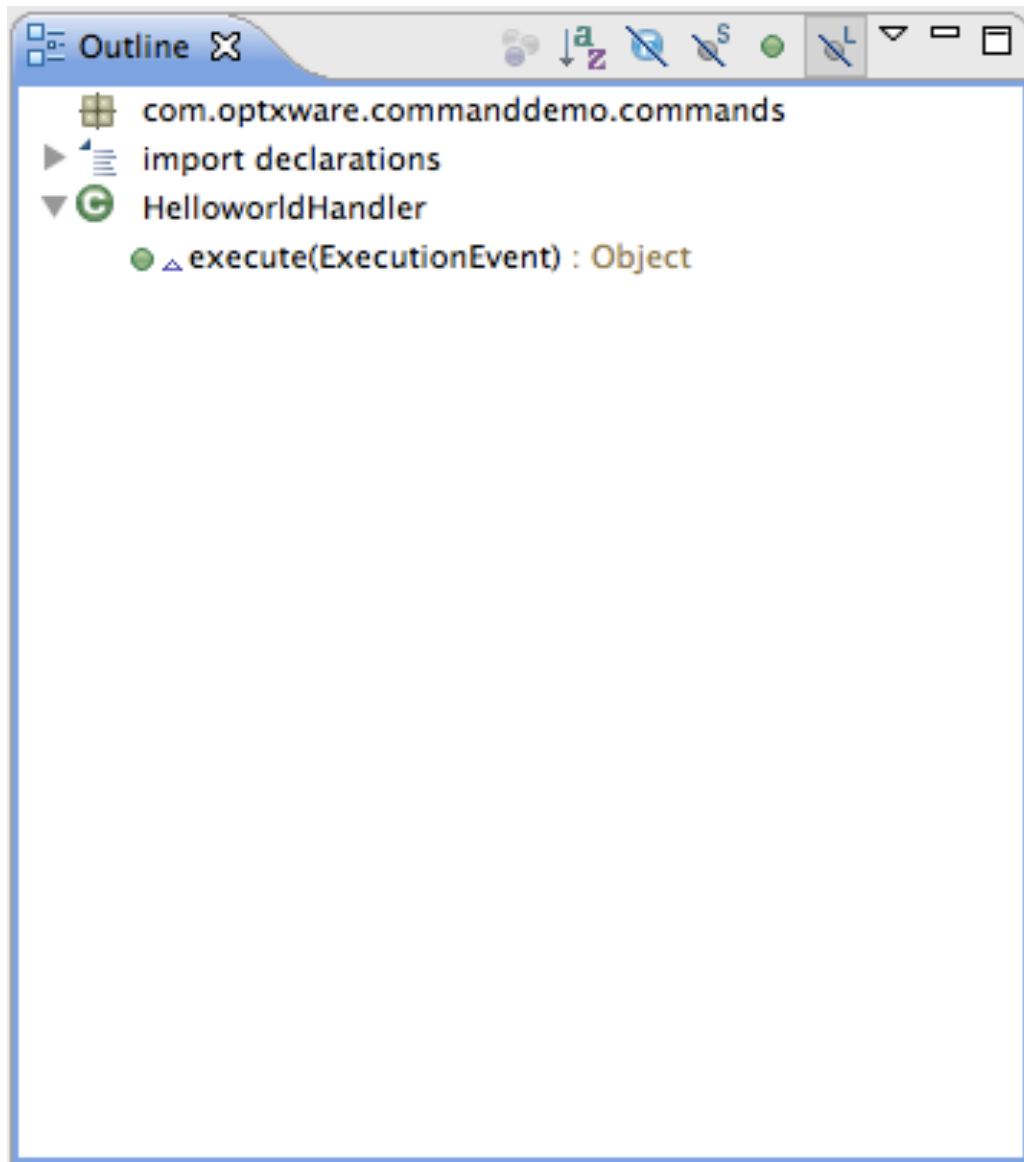
Problems @ Javadoc Declaration

0 items

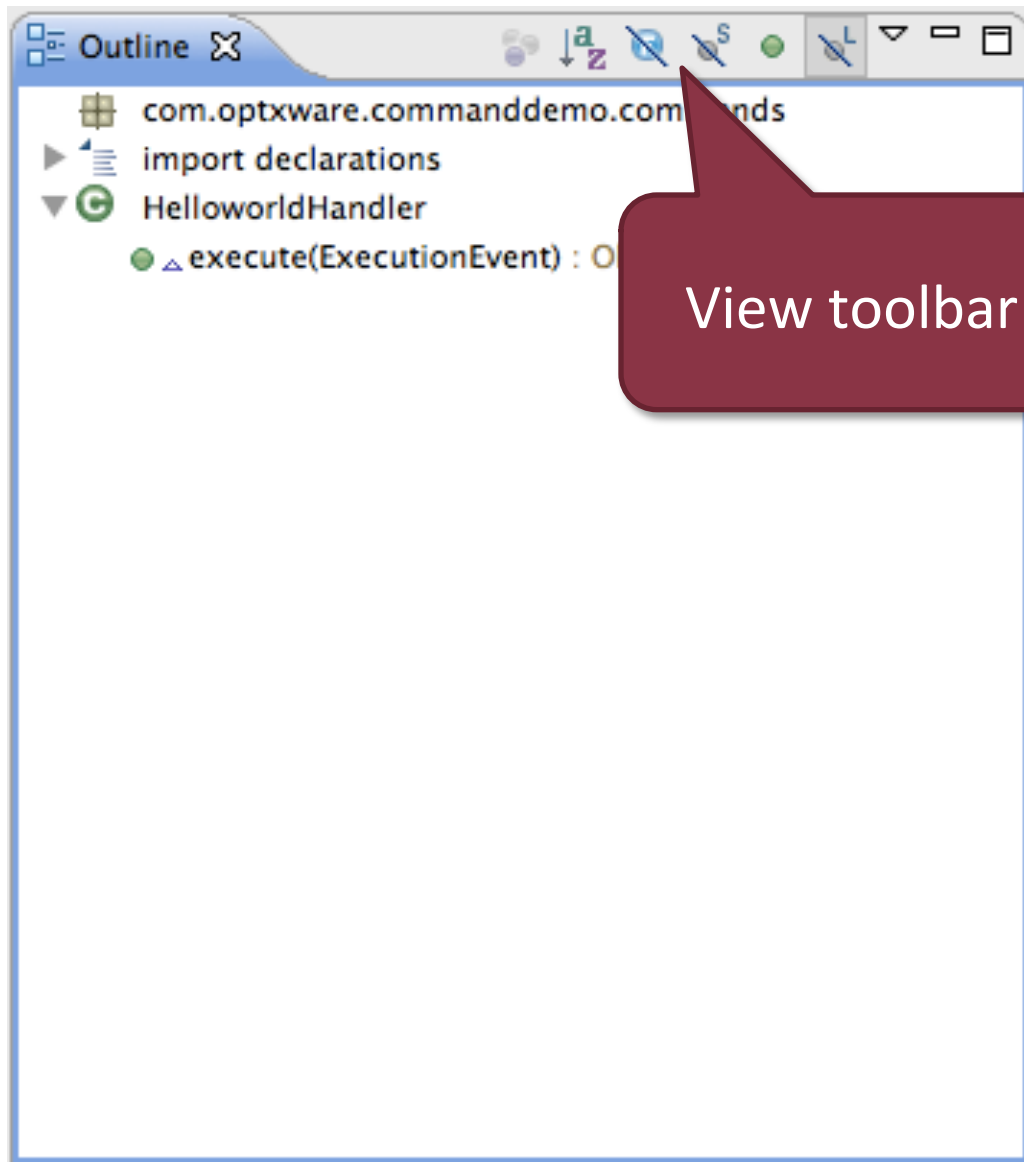
Description	Resource	Path	Location	Type

Writable Smart Insert 1 : 1

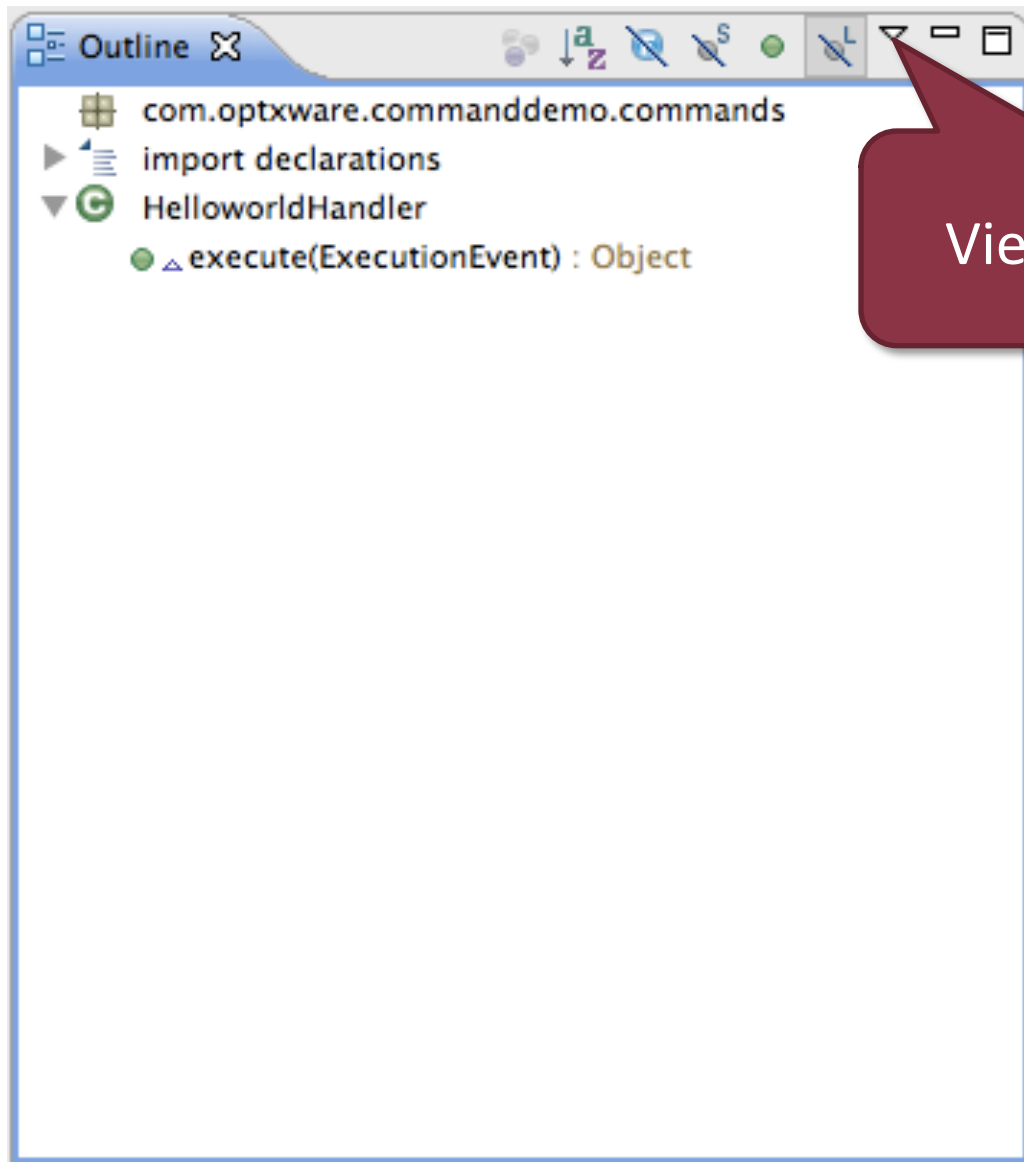
Structure of Views



Structure of Views

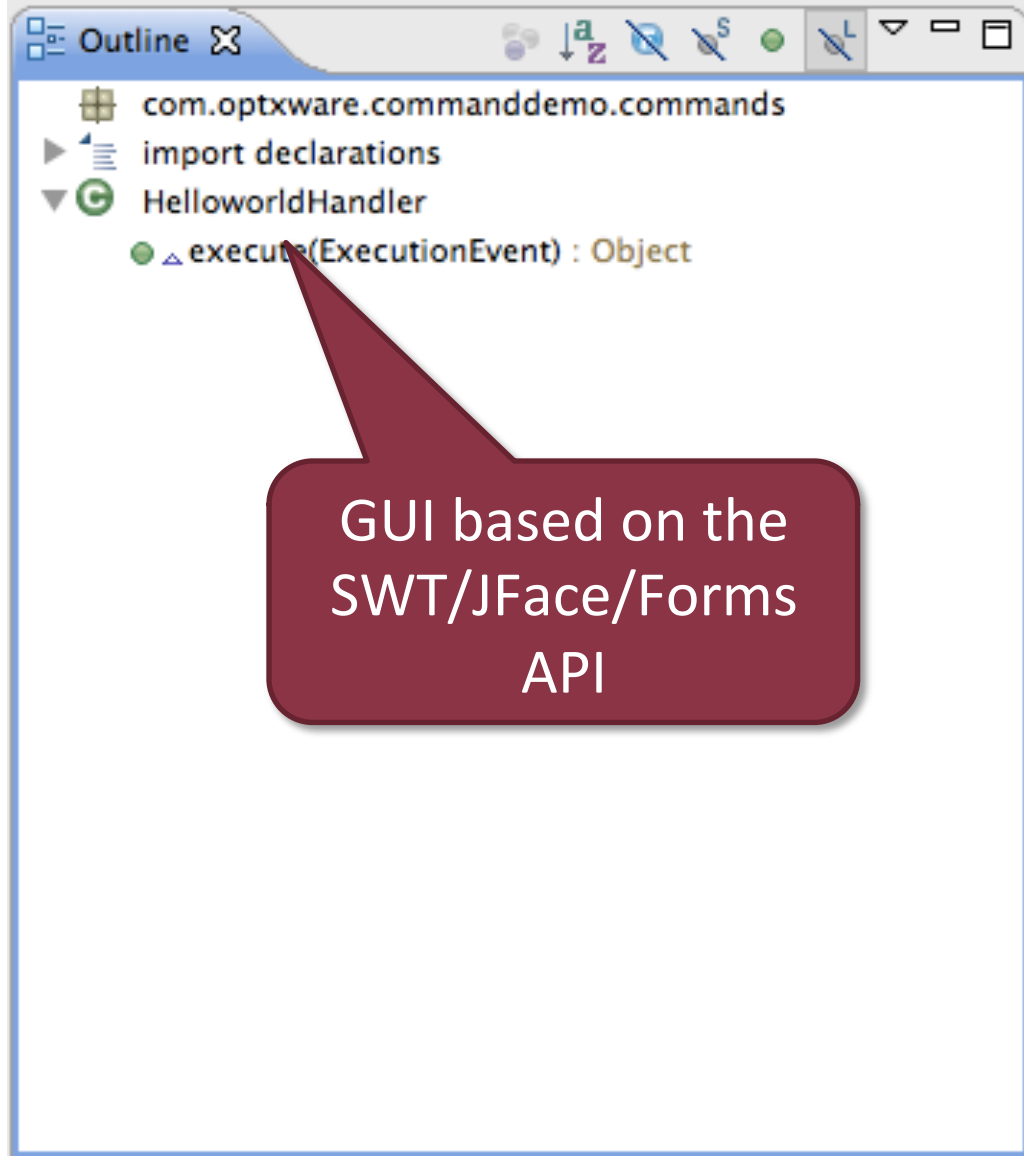


Structure of Views



View menu

Structure of Views



Views

- Helper elements for editors
 - Goal to display additional information
- May be global
 - Project Explorer
 - Error Log
- May depend on current editor/selection
 - Properties view – current selections properties
 - Outline view – quick overview of current editor contents

Implementing Views

■ View

- Extension point: `org.eclipse.ui.views`
- Interface: `IViewPart`

■ Goals

- Presenting information
- Editing information
- Handling context changes
 - E.g. selection has changed

Workbench structure

Editors (0..N)
for open files

```
package hu.optxware.eclipsecourse.rcpdemo.gui;

public interface ServiceInterface {

    public void serviceCreated();

    public void serviceRemoved();

}
```

Problems @ Javadoc Declaration

0 items

Description	Resource	Path	Location	Type

Writable

Smart Insert

1 : 1

Editor

- For editing purposes
 - Typically file editor
 - One of more files
 - But can be something else
 - contents of a database
 - compare editor
- Technologies
 - Textual (e.g. Java editor)
 - Form-based (e.g. plug-in manifest editor)
 - Graphical (e.g. UML editor)
 - Multi-page (each page may be different)

Editor registration

- Editor
 - Extension point: `org.eclipse.ui.editor`
 - Interface: `IEditorPart`
- Minimal functionality
 - Displaying editor
 - State display
 - Loading/saving data
- Building (compiling) is not the job of an editor!

Editor implementation

- `createPartControls(Composite parent)`
 - Defines the GUI elements of the editor
 - Can be any possible form
 - E.g. textual editors use a single multiline textedit component
- `init(IEditorSite site, IEditorInput input)`
 - Initializing editor
 - Loading data model
 - Presenting data model on widgets
 - Each editor must implement this

Editors – Managing Data Models

- IEditorInput: Data model handle
 - Minimal information that is required to load all contents
 - Must not contain the entire model
 - The handle remain open even after the editor closes
- Handle types
 - FileEditorInput
 - Describes a file in the workspace
 - CompareEditorInput
 - Compares and edits multiple version of file(s)
 - MultiEditorInput
 - For multi-page editors
 - Extensible with custom handle

Editors – Editor state

- `isDirty()`
 - Signals whether the contents have changed
 - If true, the *Save* command is enabled
- After editing the editor **must** send notification
 - `firePropertyChange(PROP_DIRTY)`;
 - Pull notification: new value is not pushed
 - `isDirty()` will be called in the future

Editors – Saving the Data Model

- `isSaveAsEnabled()`
 - Describes whether a copy can be made to a new file
- `doSave(IProgressMonitor monitor)`
 - Handler for the system-wide *Save* command
 - Saving is to be coded manually
 - It is possible to show save progress
- `doSaveAs()`
 - Handler for the system-wide *Save as* command
 - Save dialog is handled manually
 - Saving is coded manually
 - Monitor is missing!

Editors – Additional concepts

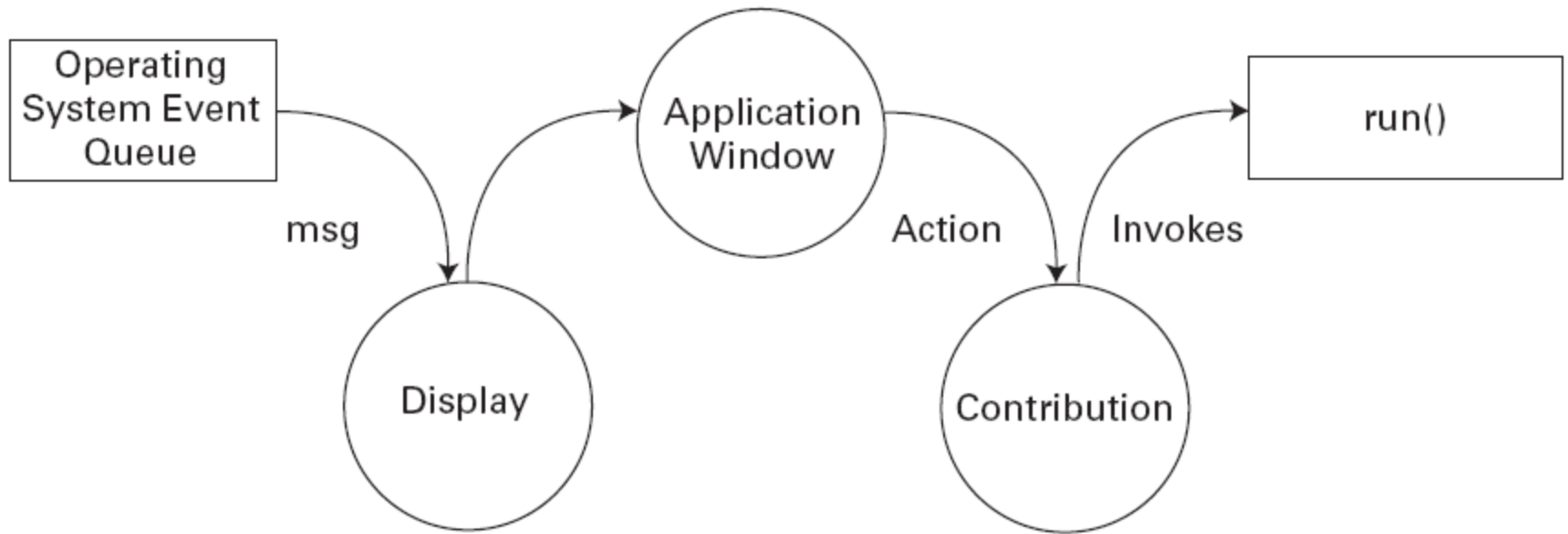
- Textual editor
 - AbstractTextEditor descendant
 - Xtext, EMFText or IMP projects
 - Higher level techniques
- Graphical editor
 - Graphical Editing Framework (GEF) project
 - Graphical Modeling Framework (GMF) project
- Multipage editors
 - MultiPageEditor descendant
 - Every page can be of a selected type

Command Framework

Actions vs Commands

- Menus, Pop-up menus and Toolbar items
 - Two approaches
 - JFace Actions
 - Older approach
 - Do not use in new projects (unless really necessary)
 - Just a very brief overview here
 - Command Framework (since Eclipse 3.3)
 - Eclipse 4.x API is based on this model

Actions



■ Actions

- Created by user interface events
- For each action contributions are registered
 - The system chooses and executes the current contribution

Action registration

- Multiple extension points (depending of goal)
 - org.eclipse.ui.editorActions
 - For editor menus
 - Interfész: IEditorActionDelegate
 - org.eclipse.ui.viewActions
 - For view menus
 - Interfész: IViewActionDelegate
 - org.eclipse.ui.popupMenus
 - For pop-up menus
 - Interfész: IEditorActionDelegate v. IViewActionDelegate v. IObjectActionDelegate

Action implementations

- `org.eclipse.jface.action.Action` descendants
 - `run()`
 - Main execution method
 - Attributes
 - Presentation: text, `toolTipText`, image, etc.
 - Behavior: enabled, checked

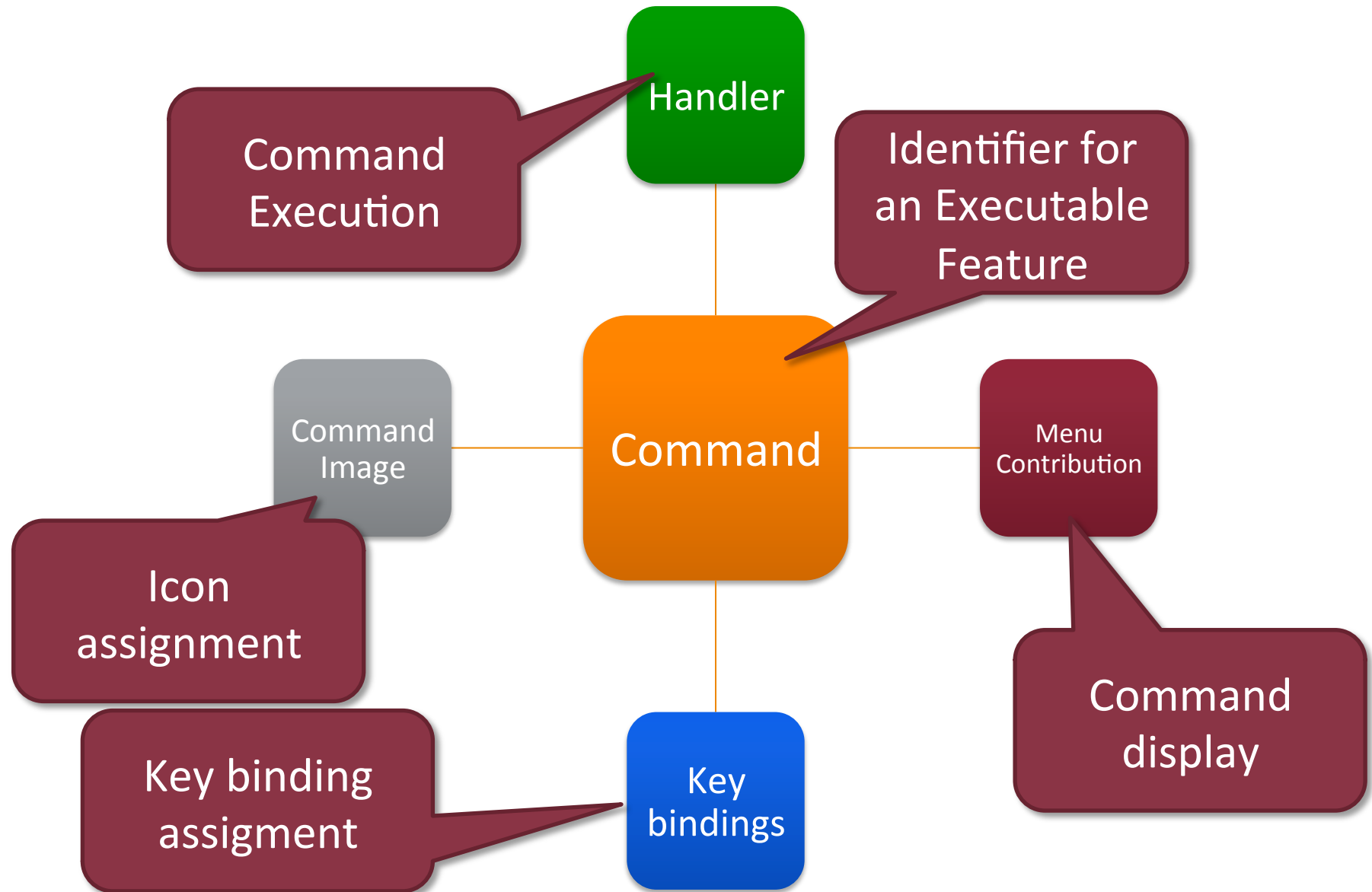
Use cases for item re-use

- Re-use supported
 - Existing command
 - Menu item/icon/binding
 - Command handler
 - What to re-use
 - Clipboard operations on view
 - Clipboard operation on editor
 - Textual
 - Graphical

Evaluation

- Multiple
 - Extension points
 - Java interfaces
- Presentation and behavior is not separated
- Problem
 - re-using Actions is problematic
 - XML duplication for registration
 - multiple Action interfaces are needed

Command Framework



Command

- Extension point: `org.eclipse.ui.commands`
- Mandatory attributes
 - Identifier
 - Name (for user interface)
- Optional attributes
 - Category
 - Default handler (not recommended to set)

Command example

```
<extension point="org.eclipse.ui.commands">  
  <command  
    description="Shows a Hello, world message."  
    id="hu.mit.bme.commanddemo.helloworld"  
    name="Display hello world">  
  </command>  
</extension>
```

Handler

- Extension point: `org.eclipse.ui.handlers`
- Mandatory parameters
 - Command identifier
 - `IHandler` or `AbstractHandler` descendant Java class
- Executable code
 - `execute(ExecutionEvent event)` method of Handler class
 - `ExecutionEvent`
 - Context information for execution
 - Use `HandlerUtil` for evaluation

Selecting the current Handler

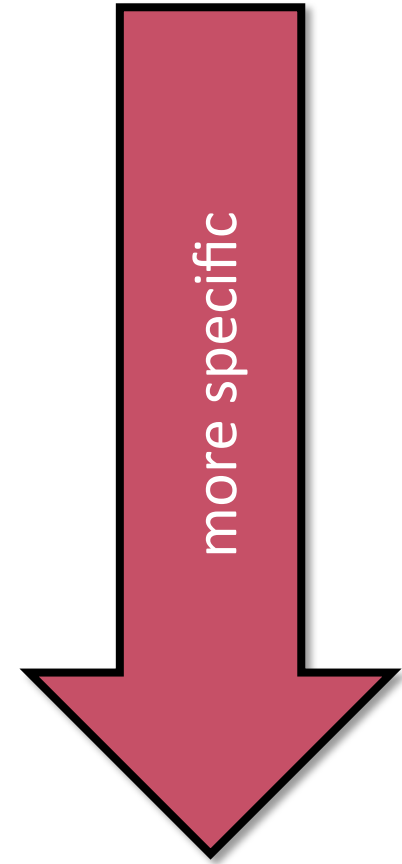
- Multiple handlers for a single command
- Choice required
 - activeWhen: activation condition (Core Expression - later)
 - enabledWhen: enablement condition (Core Expression)
- Choice algorithm:
 - All handlers with true activeWhen
 - Choose most specific condition (later)
 - If more than one handler remain, consider command disabled!

Core Expression

- Context condition setting in extensions
 - For evaluation plug-in code needs not to be loaded
- Possible conditions
 - Class checking
 - Count conditions
 - Comparisons
 - Logical operations (and, or, ...)
 - ...

Conditions on

- Active Context
- Active Action Sets
- Active Shell
- Active Workbench Window
- Active editor
- Active part
- Current selection



Example: Core Expression

- The current selection contains an IFile

```
<with variable = "selection">  
  <iterate operator ="or">  
    <instanceof  
      value =  
      "org.eclipse.core.resources.IFile">  
    </instanceof>  
  </iterate>  
</with>
```

Example: Core Expression

- The Java editor of JDT is active

```
<with variable = "activeEditorId">  
  <equals    value =  
    "org.eclipse.jdt.ui.CompilationUnitEditor">  
  </equals>  
</with>
```

Example: Handler registration

- Handler active when Java editor is active

```
<extension point="org.eclipse.ui.handlers">  
  <handler  
    class=  
      "hu.bme.mit.commanddemo.handlers.helloworld"  
    commandId="hu.bme.mit.commanddemo.helloworld">  
    <activeWhen><with variable = "activeEditorId">  
      <equals  
  
value="org.eclipse.jdt.ui.CompilationUnitEditor">  
        </equals>  
      </with></activeWhen>  
    </handler>  
  </extension>
```

Contribution

- Extension point: `org.eclipse.ui.menus`
- Describes the place of contribution
 - URL: "[Scheme]:[ID]?[Query]"
 - Scheme
 - Contribution type
 - Supported values: menu, popup, toolbar
 - ID
 - identifying the view/editor/toolbar
 - Query
 - more precise positioning inside the menu/toolbar
 - e.g. after the *Run* contribution

Contribution

■ Attributes

○ Mandatory

- Command ID

○ Optional

- Contribution ID (for query part of URL)
- Label (if not set, the command name is used)
- Icon (use CommandImages instead)

○ Visibility conditions with Core Expression

- visibleWhen child

Example: Contribution

- Toolbar icon for the main toolbar (technically a toolbar of toolbars)

```
<extension point="org.eclipse.ui.menus">
  <menuContribution      locationURI =
    "toolbar:org.eclipse.ui.main.toolbar">
    <toolbar id="hu.bme.mit.commanddemo.toolbar1">
      <command commandId=
        "hu.bme.mit.commanddemo.helloworld"
        id="hu.bme.mit.commanddemo.helloworldInToolbar"
        label="Hello, world!"
        style="push"
        tooltip="Displays a hello, world
dialog.">
      </command>
    </toolbar>
  </menuContribution>
</extension>
```

Example: Contribution

- Toolbar icon for the main toolbar (part of toolbars)

```
<extension point="org.eclipse.ui.commands">
  <menuContribution
    id="hu.bme.mit.commanddemo.menu1"
    label="Command Demo"
    toolbar="org.eclipse.ui.main.toolbar">
    <toolbar id="hu.bme.mit.commanddemo.toolbar1">
      <command commandId=
        "hu.bme.mit.commanddemo.helloworld"
        id="hu.bme.mit.commanddemo.helloworldInToolbar"
        label="Hello, world!"
        style="push"
        tooltip="Displays a hello, world
dialog.">
      </command>
    </toolbar>
  </menuContribution>
</extension>
```

URL

Example: Contribution

- Toolbar icon for the main toolbar (technically a toolbar of toolbars)

```
<extension point="org.eclipse.ui.menus">
  <menuContribution      locationURI =
    "toolbar:org.eclipse.ui.main.toolbar">
    <toolbar id="hu.bme.mit.commanddemo.toolbar1">
      <command commandId=
        "hu.bme.mit.commanddemo.helloWorld"
        id="hu.bme.mit.commanddemo.helloWorld"
        label="Hello, world"
        style="push"
        tooltip="Displays a hello, world
        dialog.">
      </command>
    </toolbar>
  </menuContribution>
</extension>
```

New toolbar
(needed on main toolbar)

Example: Contribution

- Toolbar icon for the main toolbar (technically a toolbar of toolbars)

Command ID

```
<extension point="org.eclipse.ui.menus">
  <menuContribution locationURI =
    "toolbarContribution.se.ui.main.toolbar">
    <toolbar id="hu.bme.mit.commanddemo.toolbar1">
      <command commandId=
        "hu.bme.mit.commanddemo.helloworld"
        id="hu.bme.mit.commanddemo.helloworldInToolbar"
        label="Hello, world!"
        style="push"
        tooltip="Displays a hello, world
dialog.">
      </command>
    </toolbar>
  </menuContribution>
</extension>
```

Command Icons

- Extension points: `org.eclipse.ui.commandImages`
- Mandatory attributes
 - Command ID
 - Icon
- Goal:
 - Assign icons to command
- Contribution should not contain the default icon
 - Multiple contributions of the same command!
 - Contribution might reside in a different plug-in

Example: Command Icons

```
<image
```

```
  commandId="hu.bme.mit.commanddemo.helloworld"
```

```
  icon="icons/sample.gif">
```

```
</image>
```

Key bindings

- Extension point: `org.eclipse.ui.bindings`
- Goal: assigning key bindings to commands
- Mandatory attributes:
 - Sequence: the key sequence
 - Emacs style longer sequences also supported
 - SchemeID: key binding scheme selection
 - ContextID: selecting where the binding is active
 - CommandID
- Optional attributes:
 - Platform: for platform-specific bindings
 - Locale: for language-specific bindings

Example: binding

```
<key
```

```
sequence =
```

```
"M2+F5"
```

```
commandId =
```

```
"hu.bme.mit.rcpdemo.list"
```

```
schemeId =
```

```
"org.eclipse.ui.defaultAcceleratorCo  
nfiguration"
```

```
contextId =
```

```
"org.eclipse.ui.contexts.dialog" />
```

M2: modifier key;
platform-dependent
value

Further Sources

- Eclipse Commands Tutorial
 - <http://www.vogella.de/articles/EclipseCommands/article.html>
- Eclipse Platform Architecture
 - <http://help.eclipse.org/indigo/topic/org.eclipse.platform.doc.isv/guide/arch.htm>