

La plataforma Eclipse

Curso 2013-2014

Iván Ruiz Rube
Departamento de Ingeniería Informática
Escuela Superior de Ingeniería
Universidad de Cádiz



Contenidos

- Introducción
- Proyectos
- Componentes
- Arquitectura
- Desarrollo de un proyecto Java
- Desarrollo de un plug-in
- Desarrollo de una aplicación RCP



LA PLATAFORMA ECLIPSE



INTRODUCCIÓN

No sólo un IDE...



Instalación

- Se requiere la JDK 1.6 o superior
- Acceder a <http://www.eclipse.org/>
- Entrar en *Downloads*
- Clic en *Eclipse Modeling Tools*
- Descargar versión *Win/Mac/Linux, 32/64*
- Ajustar parámetros de memoria
XX:MaxPermSize y *Xmx* en *eclipse.ini*
(*opcional*)
- Inicia *eclipse*



Eclipse IDE

- Eclipse es una plataforma de desarrollo integrada de código abierto multiplataforma.
- Eclipse fue desarrollado originalmente por IBM. Ahora es mantenido por la Fundación Eclipse.
- Participan empresas como HP, Intel, Motorola, etc.
- Dispone de una gran comunidad de usuarios.

Eclipse IDE

- Es un entorno de desarrollo de software escrito en su mayor parte en Java.
- Ideal para el desarrollo de aplicaciones web, móviles o de escritorio con Java.
- Soporte al refactoring y análisis de código.
- Integración con servidores web (Tomcat, Glassfish)
- La propia plataforma sirve para construir **aplicaciones de cliente enriquecidas.**



LA PLATAFORMA ECLIPSE



PROYECTOS



Runtimes Eclipse

- Eclipse Communication Framework
- EclipseLink Project
- Equinox
- embedded Rich Client Platform
- Gemini - Enterprise Modules Project
- Jetty - Servlet Engine and Http Server
- Rich Ajax Platform
- Riena Project
- SMILA
- Virgo



Eclipse Web Tools Platform Project

- WTP Common Tools
- Dali Java Persistence Tools
- WTP EJB Tools
- WTP Incubator
- WTP Java EE Tools
- JavaScript Development Tools
- JavaServer Faces
- Enterprise Tools for the OSGi Service Platform
- Pave
- Webtools Releng
- Server Tools
- WTP Source Editing
- Web Services Tools



SOA Platform Project

- BPEL Designer
- BPMN2 Modeler Project
- BPMN modeler
- eBAM
- eBPM
- Java Workflow Tooling
- Mangrove - SOA Modeling Framework
- SCA Tools
- Stardust
- Swordfish

Tools Project

- AJDT - AspectJ Development Tools Project
- AspectJ
- Ajax Tools Framework (ATF)
- C/C++ Development Tooling (CDT)
- WindowBuilder
- Graphical Editing Framework (GEF)
- Memory Analyzer
- Eclipse Orbit Project
- PHP Development Tools
- Parallel Tools Platform (PTP)
- Target Management

Technology Project

- Dynamic Languages Toolkit
- Eclipse Git Team Provider
- Eclipse Process Framework Project
- Eclipse Tools for Microsoft Silverlight
- The Eclipse Examples Project
- Subversive - SVN Team Provider
- Graphical Editing Framework 3D
- Hudson
- Eclipse IAM (Integration for Apache Maven)
- IDE for Education
- Linux Tools
- Open Financial Market Platform
- Open Healthcare Framework



Otros proyectos

- Eclipse Project
- Data Tools Platform
- Business Intelligence and Reporting Tools (BIRT)
- Mylyn
- Test and Performance Tools Platform Project

Eclipse Modeling Project (EMP)

The Eclipse Modeling Project focuses on the evolution and promotion of model-based development technologies within the Eclipse community by providing a unified set of modeling frameworks, tooling, and standards implementations.



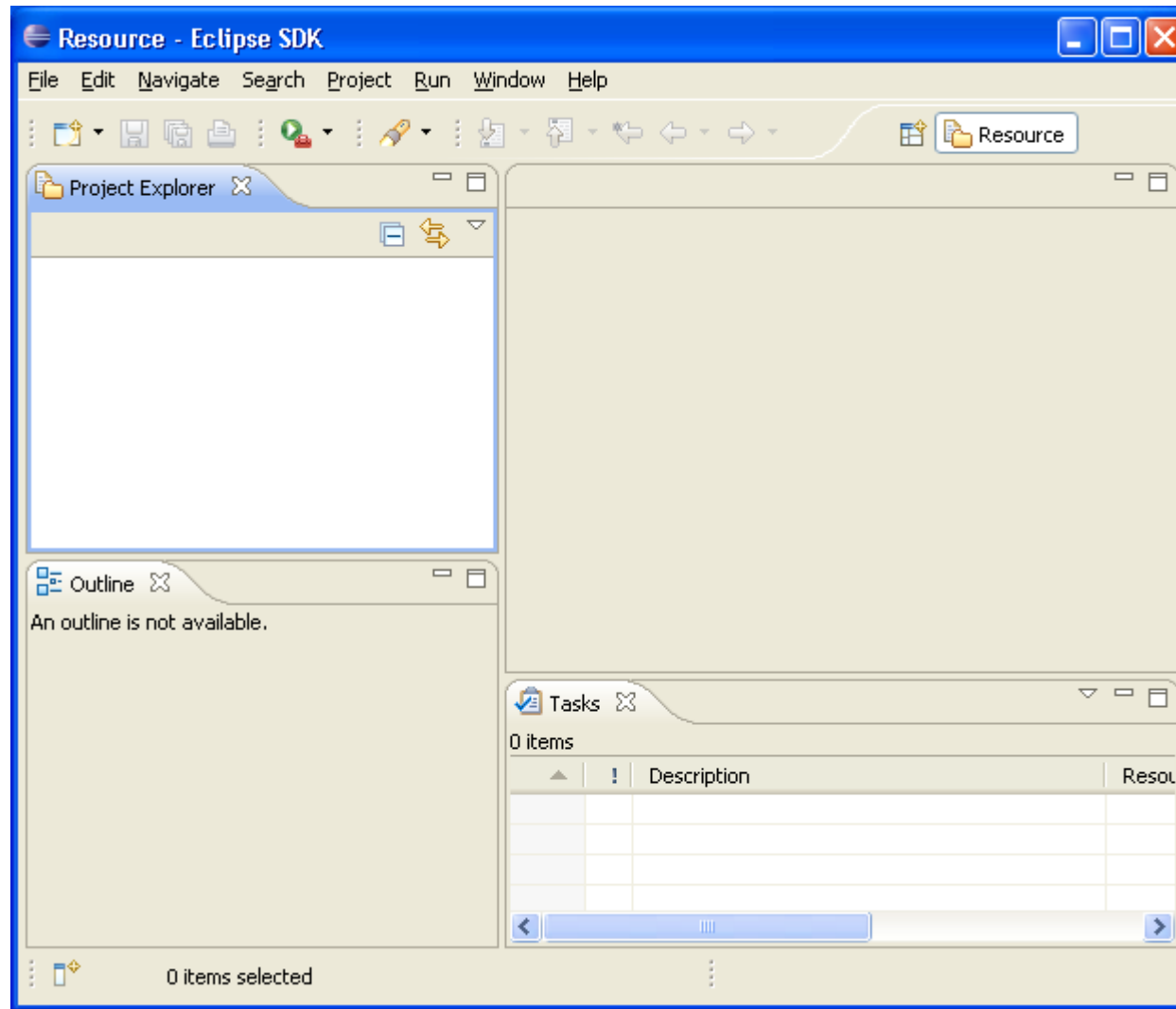


LA PLATAFORMA ECLIPSE



COMPONENTES

Workbench

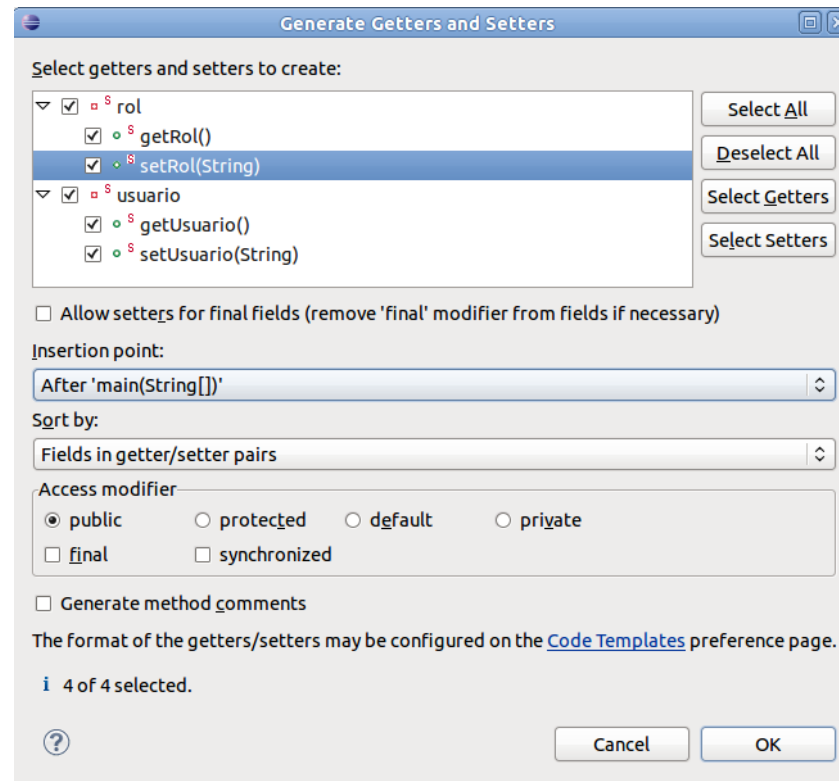




Workspace

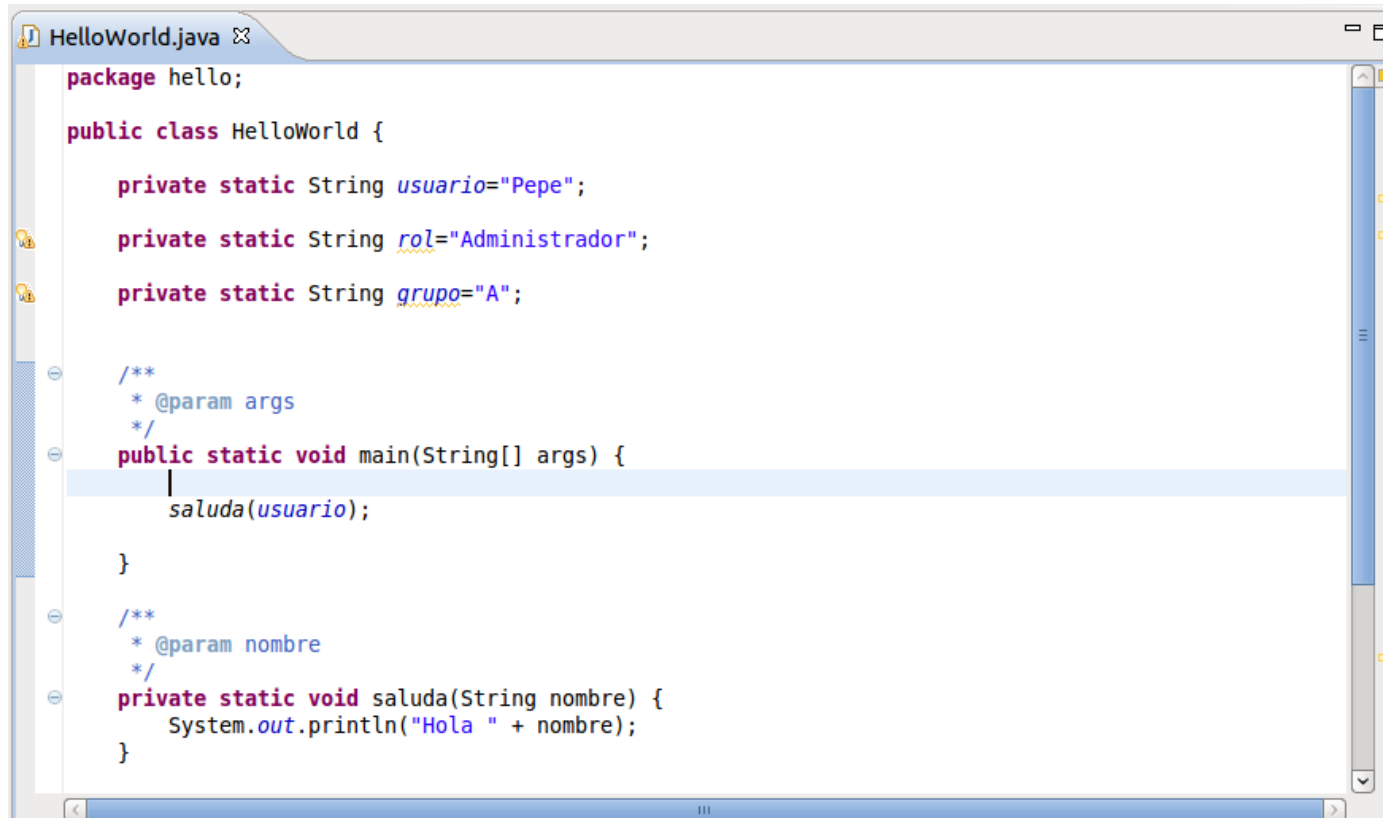
- Representa el espacio de trabajo del usuario.
- Se trata de un directorio local de la máquina donde se guardan los ficheros de trabajo.
- Utiliza metadatos (ocultos para el usuario) para llevar el histórico de modificaciones en los recursos y recordar las preferencias del usuario.
- Detecta cambios externos realizados directamente en el sistema de ficheros.

Asistentes



Los asistentes (wizards) guían al usuario a la hora de llevar a cabo un conjunto de tareas: crear un nuevo proyecto, hacer refactoring de código, etc.

Editores



```
package hello;

public class HelloWorld {

    private static String usuario="Pepe";

    private static String rol="Administrador";

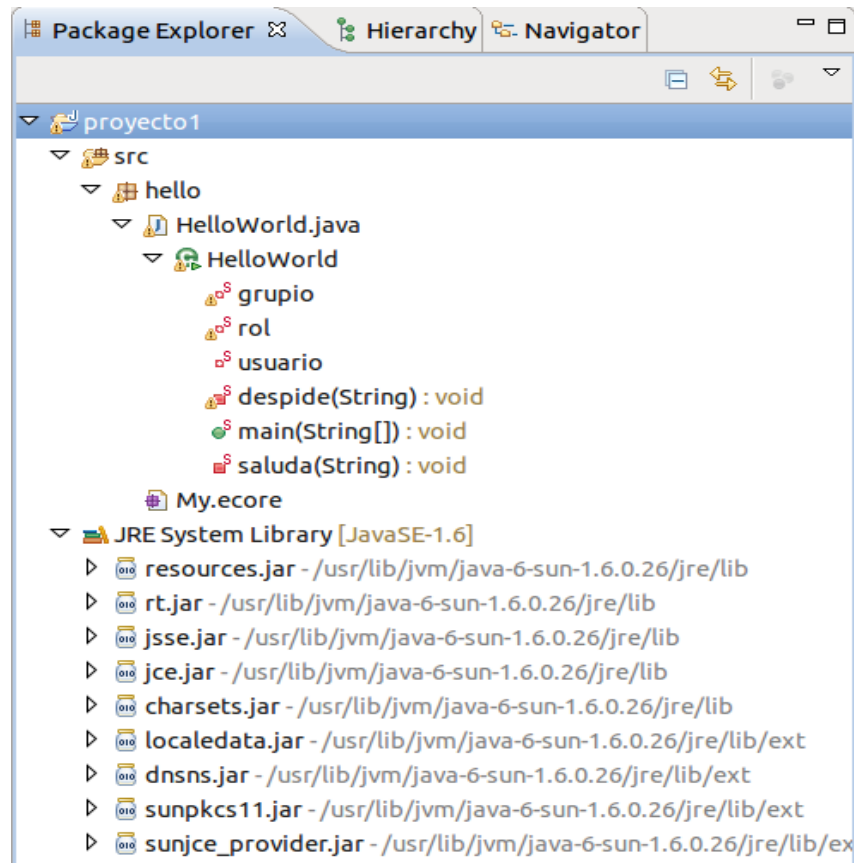
    private static String grupo="A";

    /**
     * @param args
     */
    public static void main(String[] args) {
        saluda(usuario);
    }

    /**
     * @param nombre
     */
    private static void saluda(String nombre) {
        System.out.println("Hola " + nombre);
    }
}
```

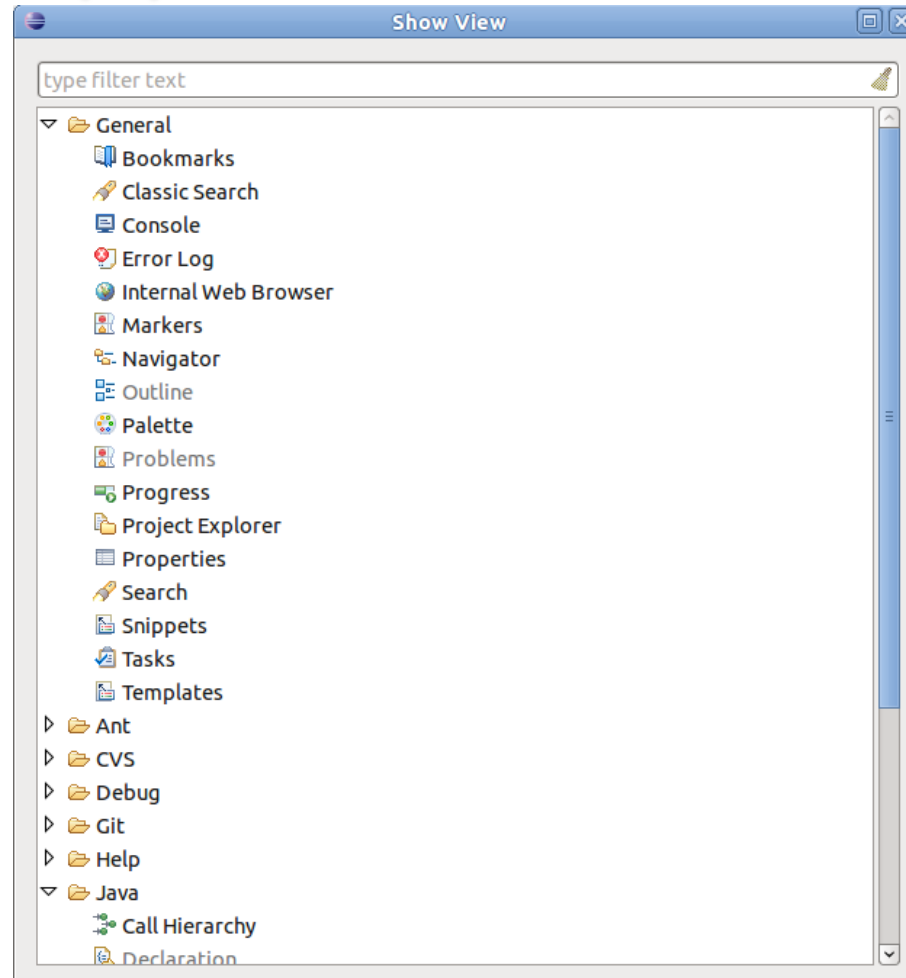
Componente visual utilizado para editar los datos de un recurso determinado. Los editores pueden ser textuales, gráficos o basados en formularios.

Vistas



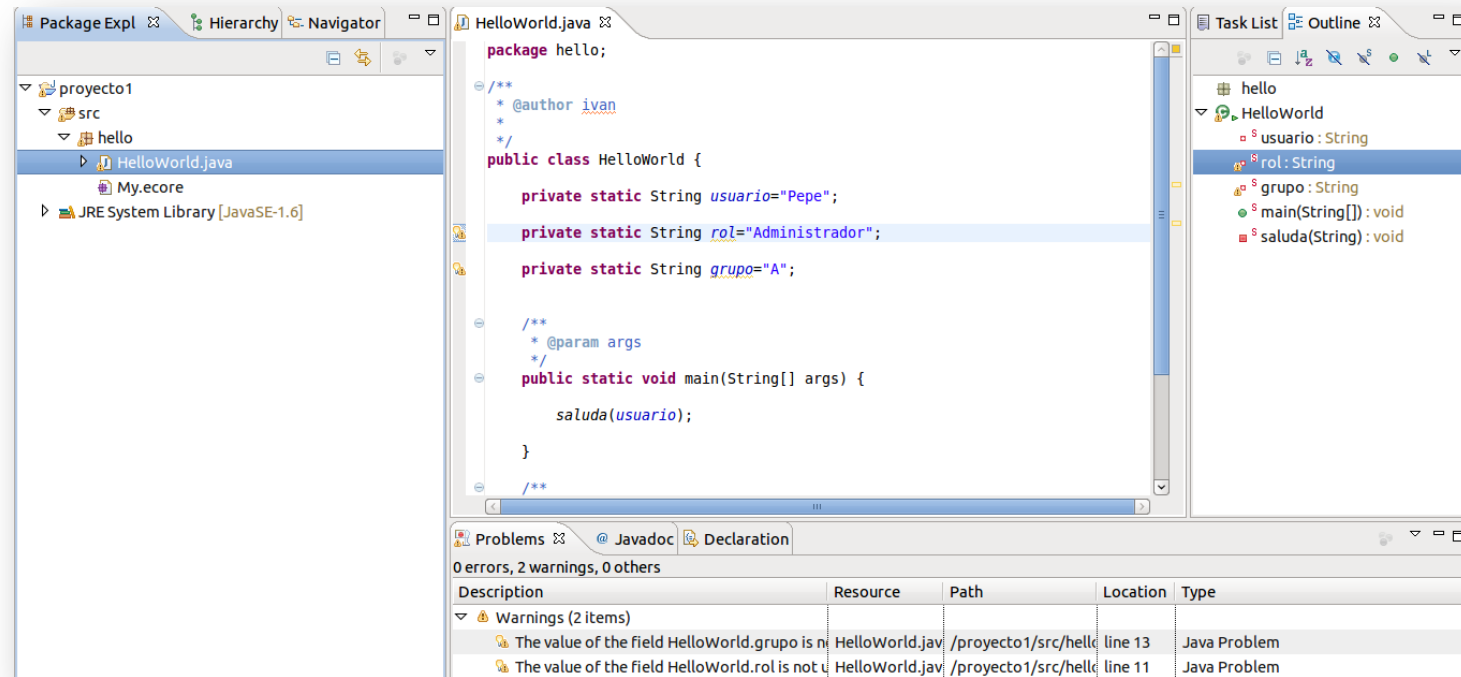
Componentes visuales que permiten navegar a través de una lista o jerarquía de elementos. También presentan propiedades específicas para para el editor activo.

Vistas (II)



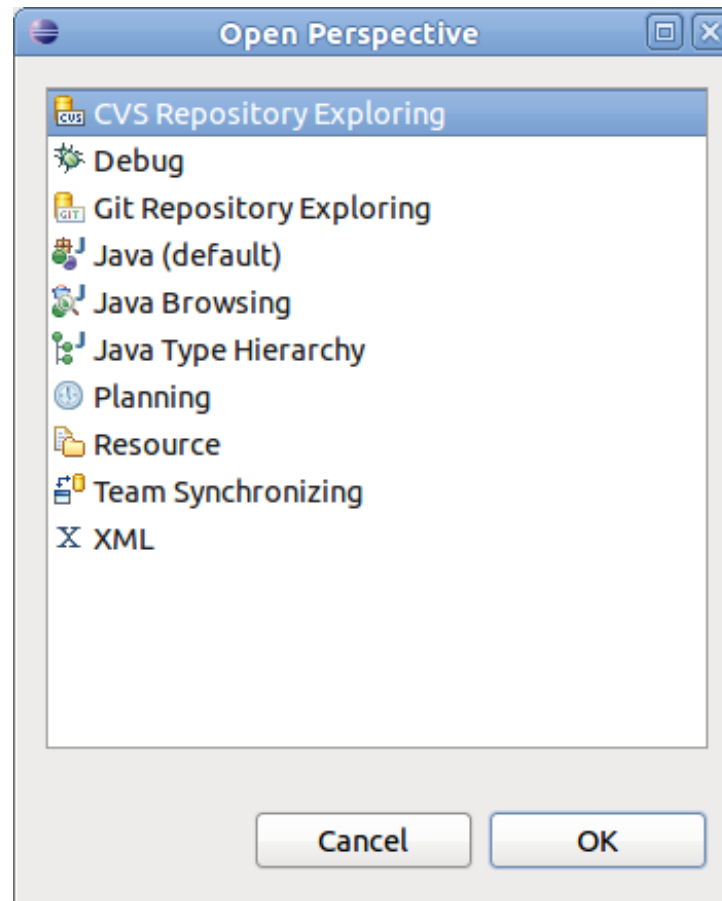
Window → Show View

Perspectivas (I)



Las perspectivas son una colección de vistas y editores

Perspectivas (II)



Window → Open Perspective

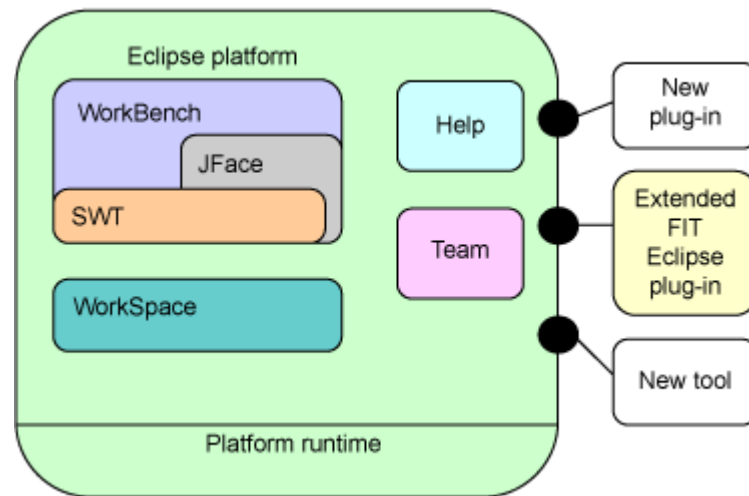


LA PLATAFORMA ECLIPSE



ARQUITECTURA

Arquitectura



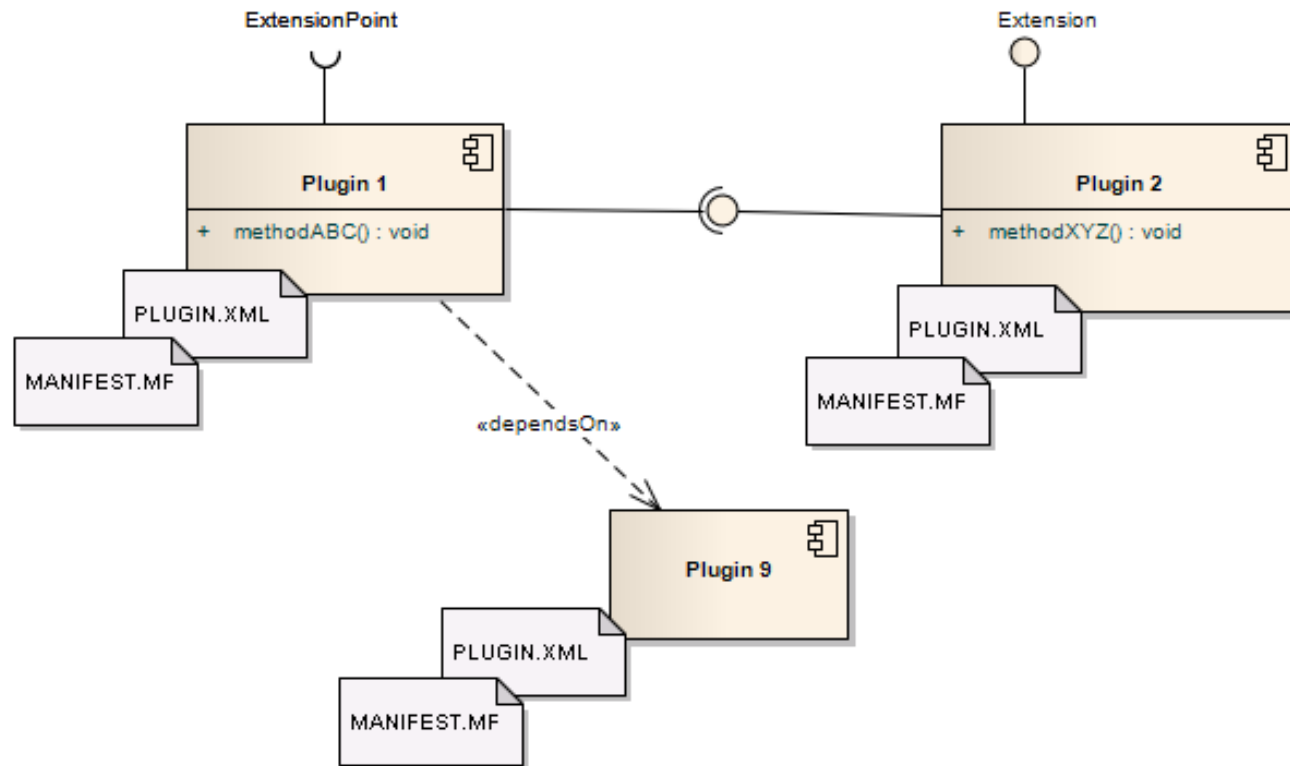
- Eclipse se compone de una base de código (kernel) y un conjunto de extensiones adicionales (plug-ins).
- La GUI de Eclipse se compone de widgets, desarrollados con SWT y utilizando un MVC con JFace.



Plug-in Development Environment (PDE)

- Eclipse permite extender la funcionalidad del IDE mediante plugins.
- Utilizando PDE podemos crear y/o editar editores, vistas, menús y asistentes del workbench de Eclipse
- Eclipse se basa en la tecnología OSGI, para proporcionar una arquitectura completamente modular.

Componentes de un Plug-in



Cada plugin define su propia API (conjunto de clases públicas), las dependencias con otros plugins, los puntos de extensión (para que otros plugins lo extiendan) y las extensiones que aportan (contribuciones a otros plugins).

Plug-ins Eclipse

- Soporte a lenguajes de programación distintos de Java: PHP, C++, Ruby, Latex, etc.
- Edición de modelos basados en lenguajes de la OMG: UML, BPMN, etc.
- Control de versiones: SVN, GIT, etc.
- Integración con sistemas de gestión de tareas: Bugzilla, Trac, JIRA, etc.
- Etc.



Eclipse para todos

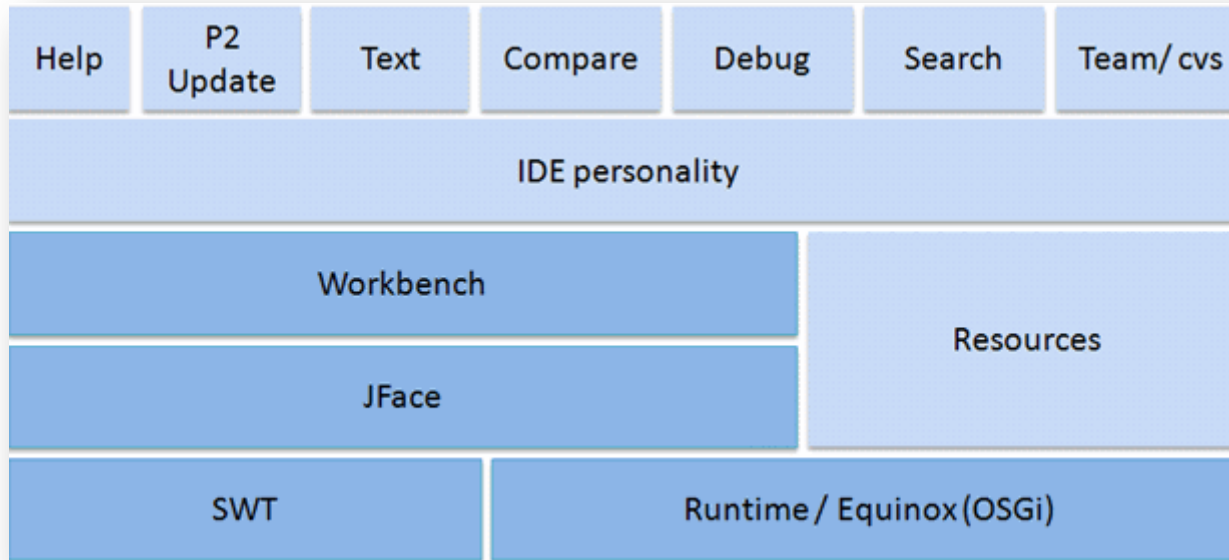
- Eclipse IDE for Java EE Developers
- Eclipse IDE for JS Web Developers
- Eclipse for RCP and RAP Developers
- Eclipse IDE for C/C++ Linux Developers
- Eclipse for Testers
- Eclipse IDE for Parallel Application Developers
- **Eclipse Modeling Tools**
- ...



Rich Client Platform (RCP)

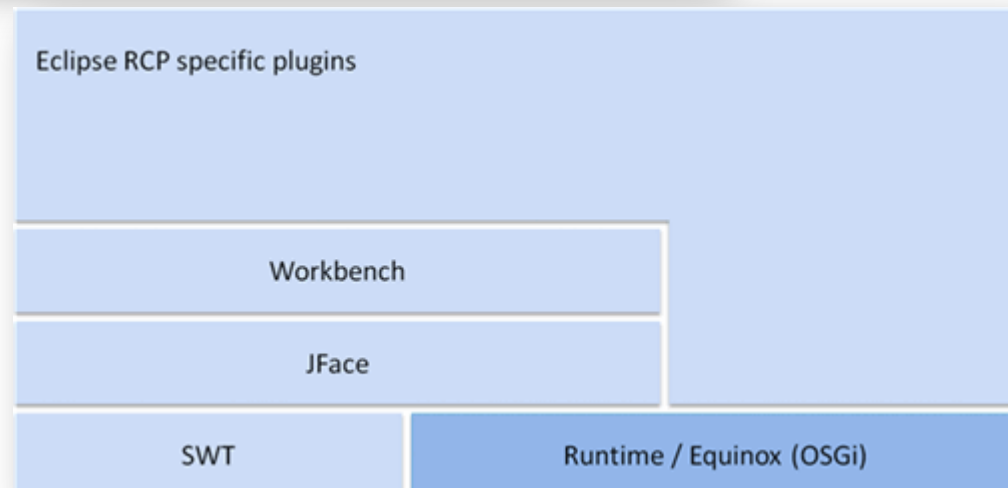
- Desde la versión 3.0, podemos utilizar la propia plataforma para crear aplicaciones de escritorio extensibles.
- Estas aplicaciones (RCP) se basan en el kernel de Eclipse y un conjunto de plugins seleccionados o desarrollados.
- Se distribuyen como aplicaciones independientes.

Eclipse IDE y Aplicaciones RCP



Eclipse IDE

Aplicación RCP



* Lars Vogel



Aplicaciones basadas en Eclipse

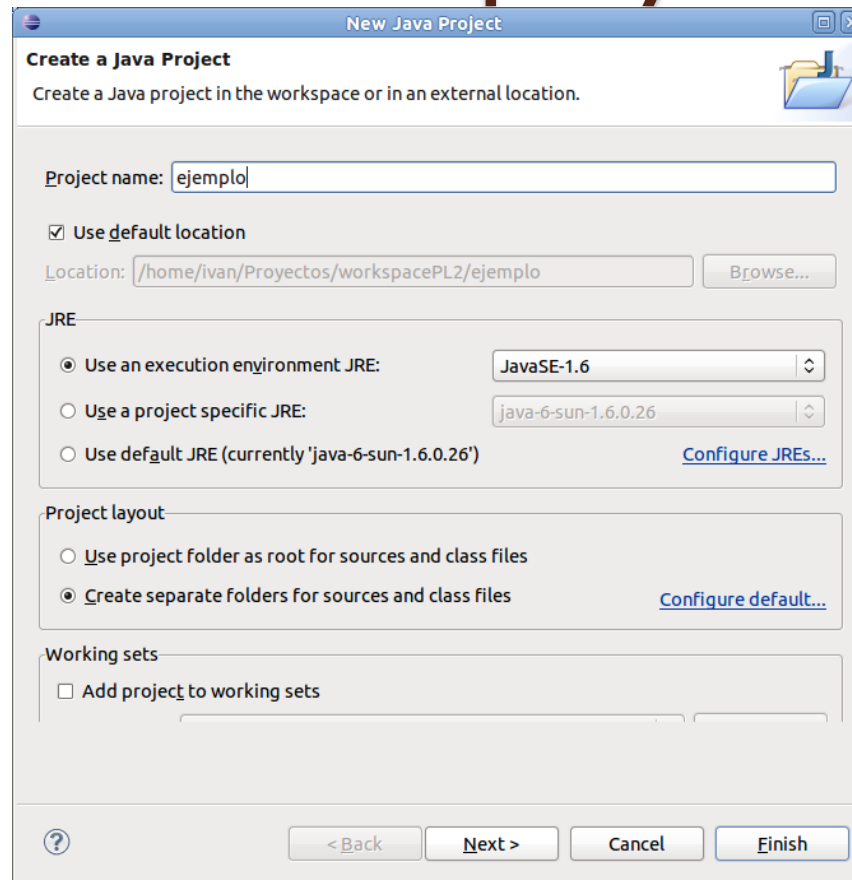
- Adobe Flex Builder
- IBM Rational Method Composer,
- MyEclipse
- IBM Lotus Notes 8
- SpringSource Tool Suite
- Etc.

LA PLATAFORMA ECLIPSE



DESARROLLO DE UN PROYECTO JAVA

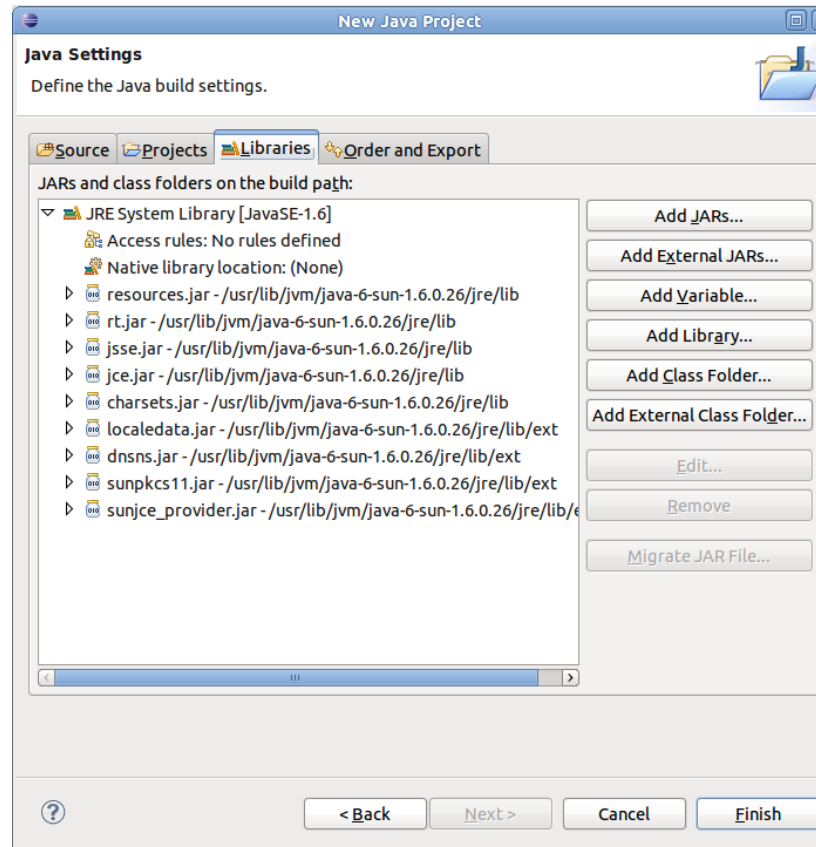
Creación de un proyecto Java



File → New → Java Project

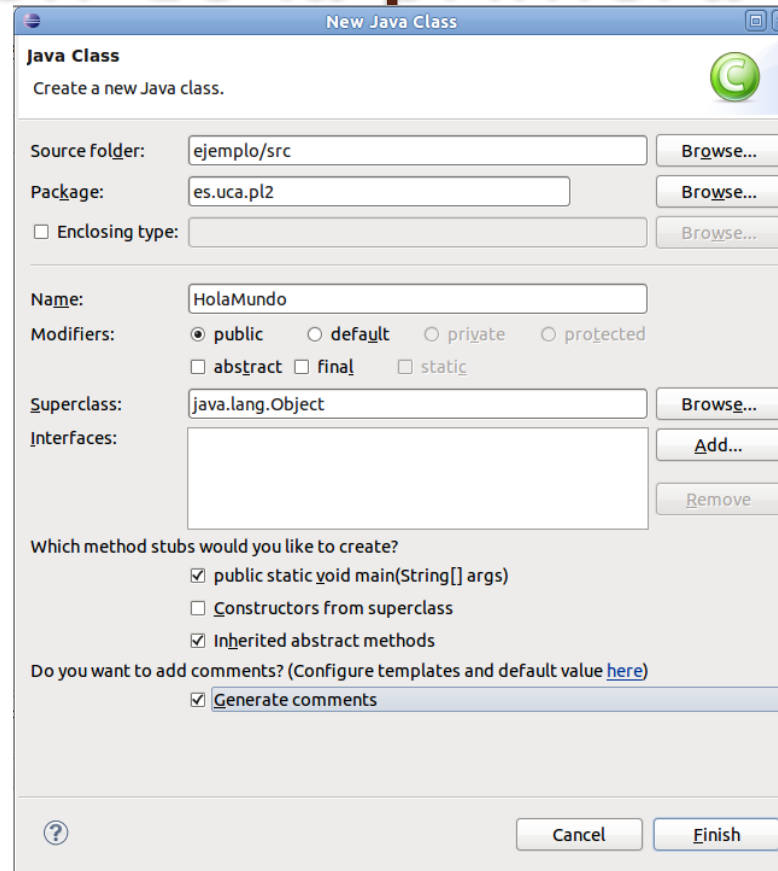
Es preciso definir, al menos, el nombre del proyecto y el entorno de ejecución de Java (JRE o JDK)

Configuración del proyecto



Estableceremos la carpeta del código fuente (*Source*), las dependencias con otros proyectos del workspace (*Projects*) y con JARs externos (*Libraries*) y el orden de compilación (*Order and Export*)

Creación de la primera clase Java



File → New → Class

Definimos *nombre* de la clase, *paquete* donde se almacenará, *modificadores* de acceso, *superclases*, *interfaces* a implementar y si queremos añadir *method stubs* y *comentarios*

Escribiendo código

```

HolaMundo.java
/**
 * Mi primer programa en Eclipse
 */
package es.uca.pl2;

/**
 * Clase de ejemplo Hola Mundo
 */
public class HolaMundo {

    /**
     * Método principal: Hola Mundo
     * @param args
     */
    public static void main(String[] args) {
        // TODO Auto-generated method stub
        System.out.println("Hola Mundo!!");
    }
}

```

Problems | @ Javadoc | Declaration | Console

1 error, 0 warnings, 0 others

Description	Resource	Location	Type
Errors (1 item)			
Syntax error, insert ";" to complete BlockStatements	HolaMundo.java	line 18	Java Problem

El editor de código de Eclipse ofrece autocompletado, quick fixes, coloreado de sintaxis e inclusión de plantillas de código y comentarios (JavaDoc), entre otras capacidades.

Mejorando el código

Toggle Comment	Shift+Ctrl+C
Remove Block Comment	Shift+Ctrl+\
Generate Element Comment	Shift+Alt+J
Correct Indentation	Ctrl+I
Format	Shift+Ctrl+F
Format Element	
Add Import	Shift+Ctrl+M
Organize Imports	Shift+Ctrl+O
Sort Members...	
Clean Up...	
Override/Implement Methods...	
Generate Getters and Setters...	
Generate Delegate Methods...	
Generate hashCode() and equals()...	
Generate toString()...	
Generate Constructor using Fields...	
Generate Constructors from Superclass...	
Externalize Strings...	

Source

Rename...	Shift+Alt+R
Move...	Shift+Alt+V
Change Method Signature...	Shift+Alt+C
Inline...	Shift+Alt+I
Extract Interface...	
Extract Superclass...	
Use Supertype Where Possible...	
Pull Up...	
Push Down...	
Extract Class...	
Introduce Parameter Object...	
Introduce Indirection...	
Infer Generic Type Arguments...	

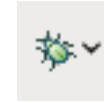
Refactoring

Compilación y ejecución



- Eclipse compila “al vuelo” los ficheros fuente que estamos modificando.
- Para ejecutar, tenemos que seleccionar la clase con el método *main* deseado y luego *Run As* → *Java Application* en el menú contextual (botón derecho).
- La vista *Console*, muestra la salida de la aplicación

Depuración de código



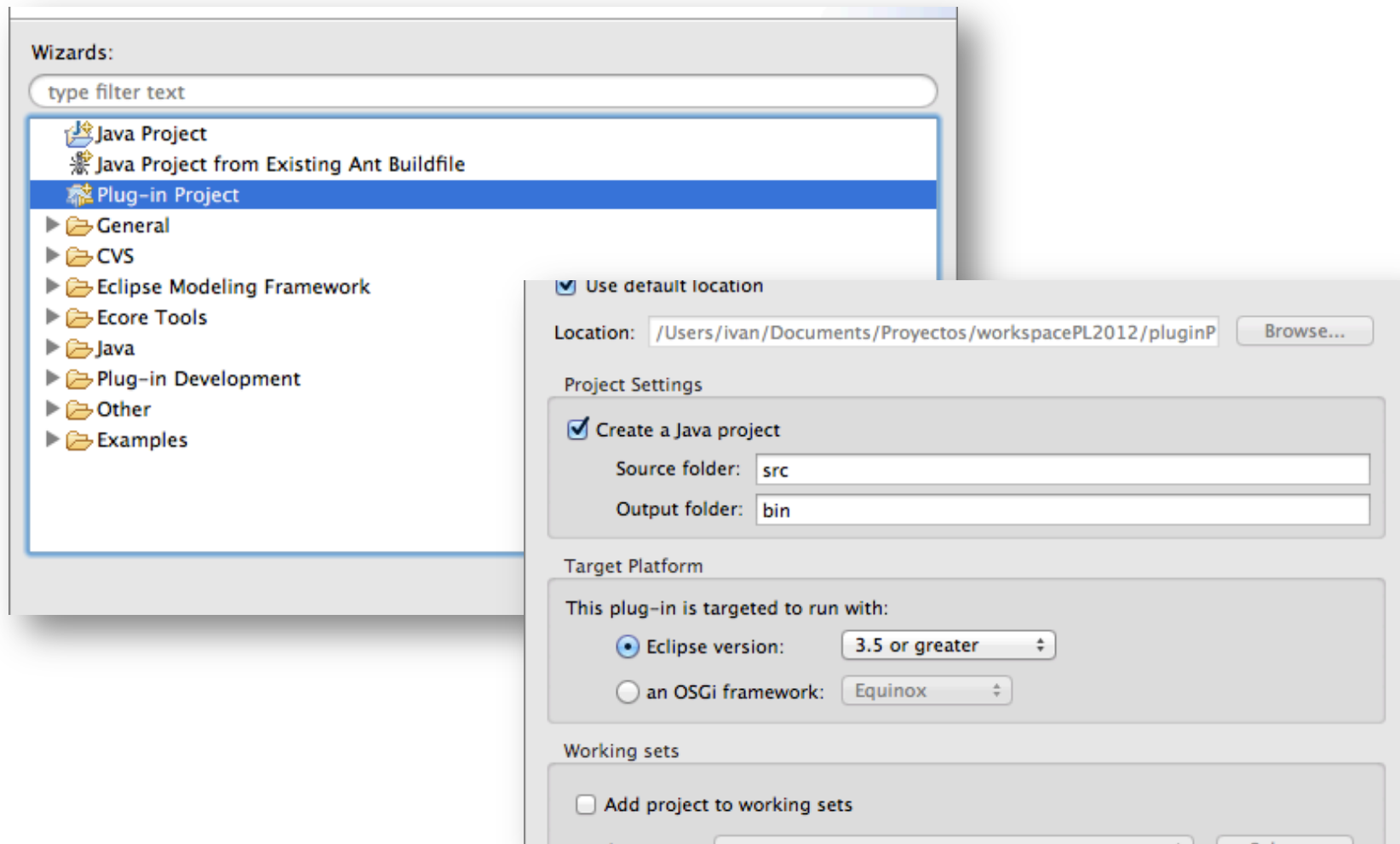
- Eclipse ofrece una completa perspectiva de depuración, incluyendo:
 - Sesión de depuración
 - Puntos de ruptura (breakpoints)
 - Inspección de variables/expresiones
- Control de la ejecución
 - *Debug*: inicia la depuración
 - *Step into*: ejecuta la instrucción, entrando en métodos
 - *Step over*: ejecuta la instrucción, sin entrar en métodos
 - *Step return*: ejecuta hasta final del método actual
 - *Pause*: detiene la ejecución
 - *Resume*: continúa con la ejecución de la aplicación
 - *Terminate*: finaliza el proceso

LA PLATAFORMA ECLIPSE

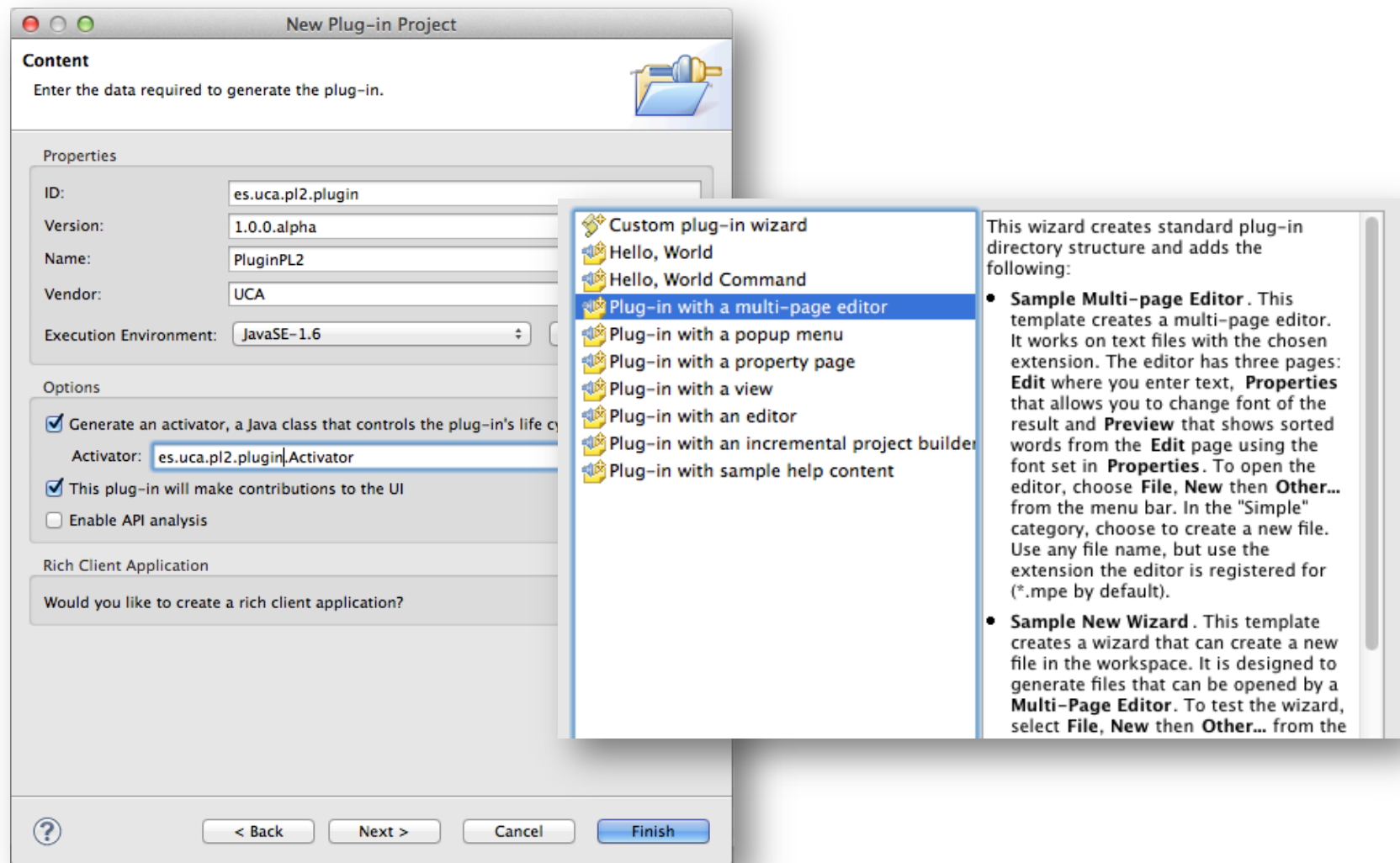


DESARROLLO DE UN PLUG-IN

Creación de un plug-in



Creación de un plug-in (II)



Content
Enter the data required to generate the plug-in.

Properties

ID: es.uca.pl2.plugin
Version: 1.0.0.alpha
Name: PluginPL2
Vendor: UCA
Execution Environment: JavaSE-1.6

Options

Generate an activator, a Java class that controls the plug-in's life cycle
Activator: es.uca.pl2.plugin.Activator
 This plug-in will make contributions to the UI
 Enable API analysis

Rich Client Application

Would you like to create a rich client application?

- Custom plug-in wizard
- Hello, World
- Hello, World Command
- Plug-in with a multi-page editor**
- Plug-in with a popup menu
- Plug-in with a property page
- Plug-in with a view
- Plug-in with an editor
- Plug-in with an incremental project builder
- Plug-in with sample help content

This wizard creates standard plug-in directory structure and adds the following:

- **Sample Multi-page Editor**. This template creates a multi-page editor. It works on text files with the chosen extension. The editor has three pages: **Edit** where you enter text, **Properties** that allows you to change font of the result and **Preview** that shows sorted words from the **Edit** page using the font set in **Properties**. To open the editor, choose **File, New** then **Other...** from the menu bar. In the "Simple" category, choose to create a new file. Use any file name, but use the extension the editor is registered for (*.mpe by default).
- **Sample New Wizard**. This template creates a wizard that can create a new file in the workspace. It is designed to generate files that can be opened by a **Multi-Page Editor**. To test the wizard, select **File, New** then **Other...** from the

Creación de un plug-in (III)

The image shows two overlapping dialog boxes from the Eclipse IDE. The background dialog is titled "Sample Multi-Page Editor" and contains the following fields:

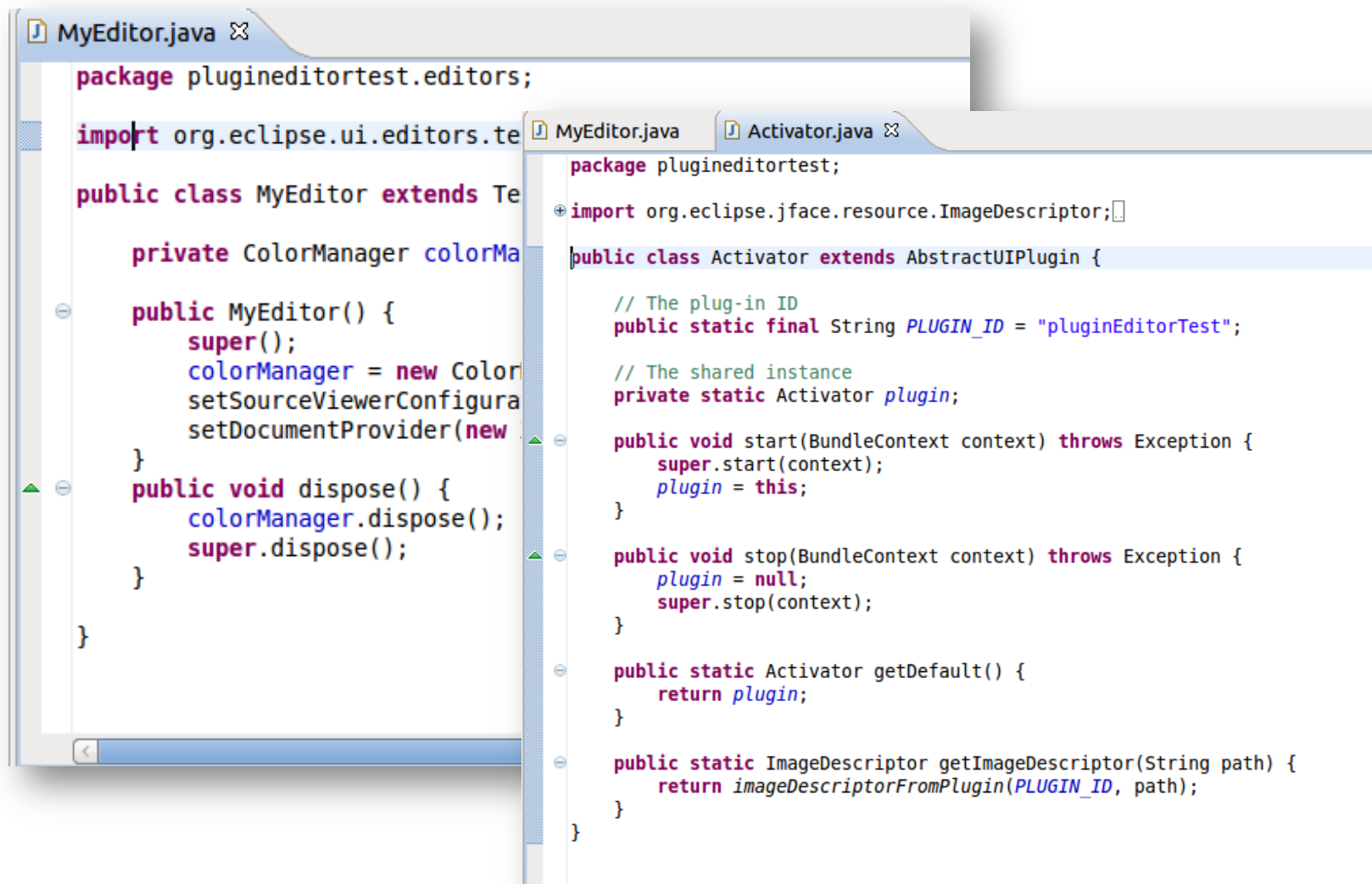
- Java Package Name: `es.uca.pl2.plugin.editors`
- Editor Class Name: `MultiPageEditor`
- Editor Contributor Class Name: `MultiPageEditorContributor`
- Editor Name: `Sample Multi-page Editor`
- File Extension: `mpe`

The foreground dialog is titled "New Wizard Options" and contains the following fields:

- Java Package Name: `es.uca.pl2.plugin.wizards`
- Wizard Category ID: `es.uca.pl2.plugin`
- Wizard Category Name: `Sample Wizards`
- Wizard Class Name: `SampleNewWizard`
- Wizard Page Class Name: `SampleNewWizardPage`
- Wizard Name: `Multi-page Editor file PL2`
- File Extension: `mpe`
- Initial File Name: `new_file.mpe`

Both dialogs have a "? Help" icon and navigation buttons: "< Back", "Next >", "Cancel", and "Finish".

Creación de un plug-in (IV)



```
MyEditor.java
package plugineditor.test.editors;

import org.eclipse.ui.editors.text.*;

public class MyEditor extends TextEditor {

    private ColorManager colorManager;

    public MyEditor() {
        super();
        colorManager = new ColorManager();
        setSourceViewerConfigurer(new ColorSourceViewerConfigurer());
        setDocumentProvider(new ColorDocumentProvider());
    }

    public void dispose() {
        colorManager.dispose();
        super.dispose();
    }

}

Activator.java
package plugineditor.test;

import org.eclipse.jface.resource.ImageDescriptor;

public class Activator extends AbstractUIPlugin {

    // The plug-in ID
    public static final String PLUGIN_ID = "pluginEditorTest";

    // The shared instance
    private static Activator plugin;

    public void start(BundleContext context) throws Exception {
        super.start(context);
        plugin = this;
    }

    public void stop(BundleContext context) throws Exception {
        plugin = null;
        super.stop(context);
    }

    public static Activator getDefault() {
        return plugin;
    }

    public static ImageDescriptor getImageDescriptor(String path) {
        return imageDescriptorFromPlugin(PLUGIN_ID, path);
    }

}
```

Test del plug-in

es.uca.pl2.plugin

Overview

General Information

This section describes general information about this plug-in.

ID:

Version:

Name:

Vendor:

Platform Filter:

Activator:

Activate this plug-in when one of its classes is loaded

This plug-in is a singleton

Plug-in Content

The content of the plug-in is made up of two sections:

- [Dependencies](#): lists all the plug-ins required on this plug-in's classpath to compile and run.
- [Runtime](#): lists the libraries that make up this plug-in's runtime.

Extension / Extension Point Content

This plug-in may define extensions and extension points:

- [Extensions](#): declares contributions this plug-in makes to the platform.
- [Extension Points](#): declares new function points this plug-in adds to the platform.

Execution Environments

Specify the minimum execution environments required to run this plug-in.

JavaSE-1.6

Testing

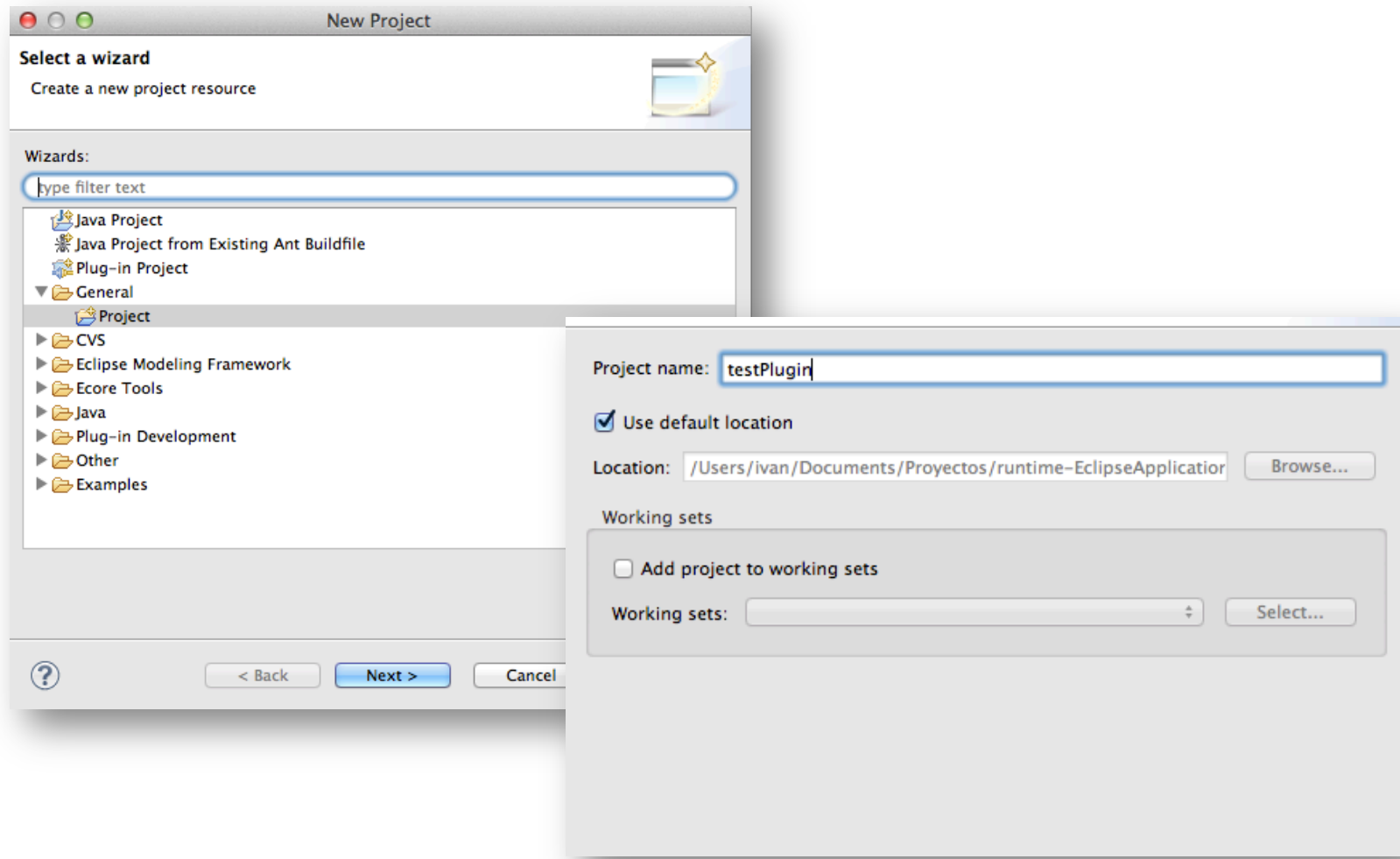
Test this plug-in by launching a separate Eclipse application:

- [Launch an Eclipse application](#)
- [Launch an Eclipse application in Debug mode](#)

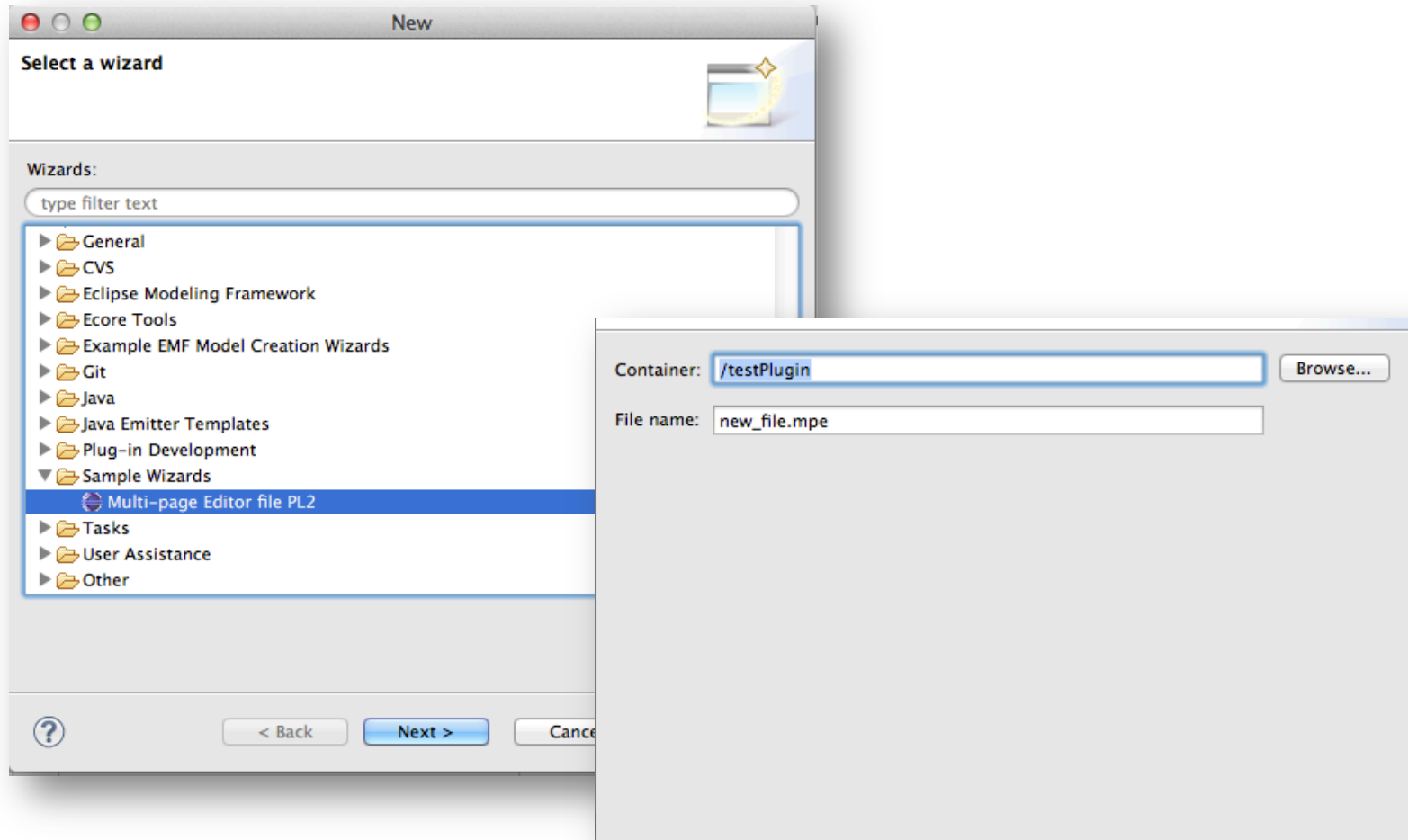
Exporting

Overview | Dependencies | Runtime | Extensions | Extension Points | Build | MANIFEST.MF | plugin.xml | build.properties

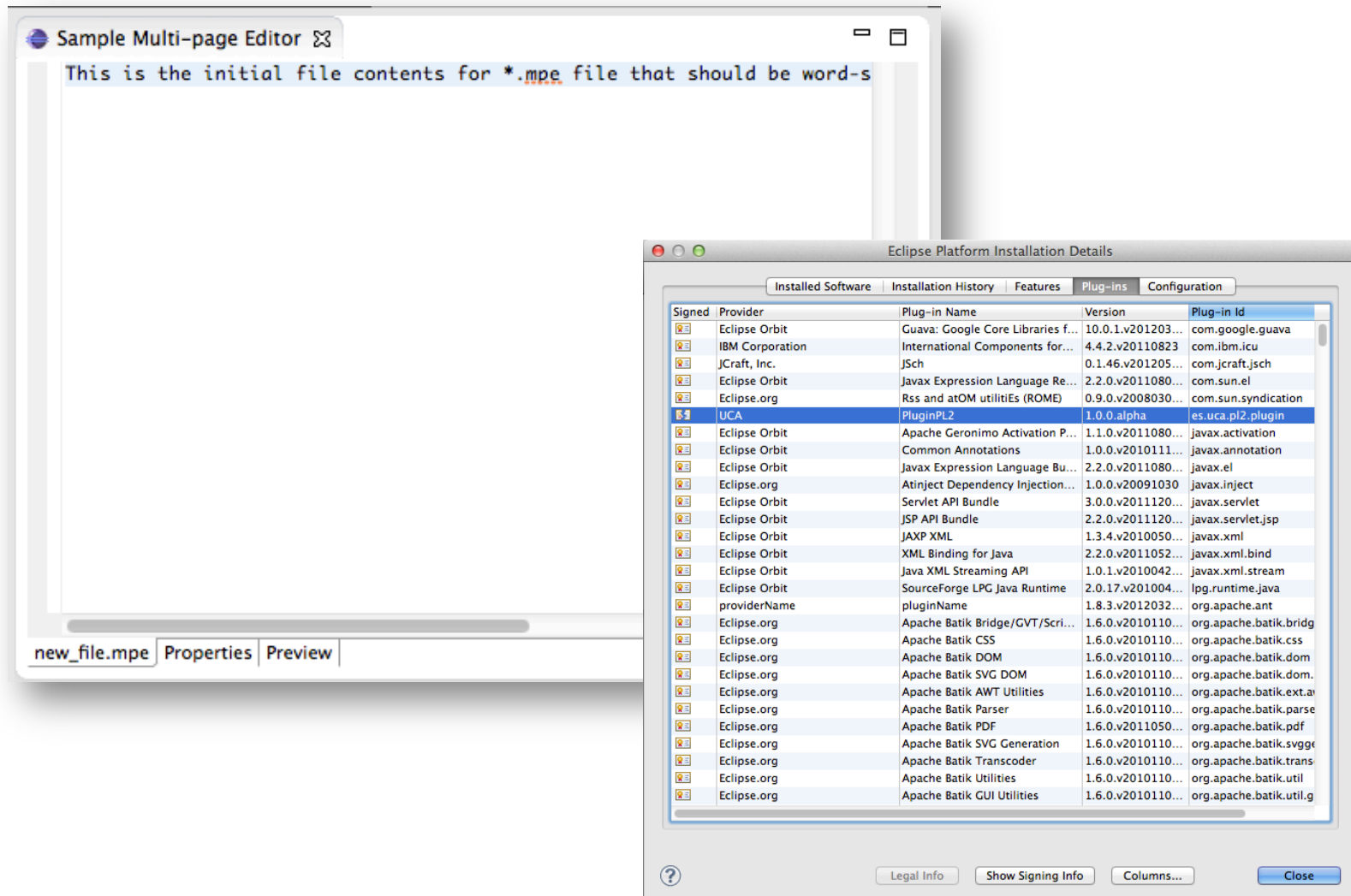
Test del plug-in (II)



Test del plug-in (III)



Test del plug-in (IV)



LA PLATAFORMA ECLIPSE



DESARROLLO DE UNA APLICACIÓN RCP

Creación de una aplicación RCP

Plug-in Project
Create a new plug-in project

Project name: applicationPL2

Use default location

Location: /Users/ivan/Documents/Proyectos/workspacePL2012/applcal Browse...

Project Settings

Create a Java project

Source folder: src

Output folder: bin

Target Platform

This plug-in is targeted to run with:

Eclipse version: 3.5 or greater

an OSGi framework: Equinox

Working sets

Add project to working sets

Working sets: Select...

< Back Next > Cancel Finish

Content
Enter the data required to generate the plug-in.

Properties

ID: es.uca.pl2.application

Version: 1.0.0.alpha

Name: ApplicationPL2

Vendor: UCA

Execution Environment: JavaSE-1.6 Environments...

Options

Generate an activator, a Java class that controls the plug-in's life cycle

Activator: es.uca.pl2.application.Activator

This plug-in will make contributions to the UI

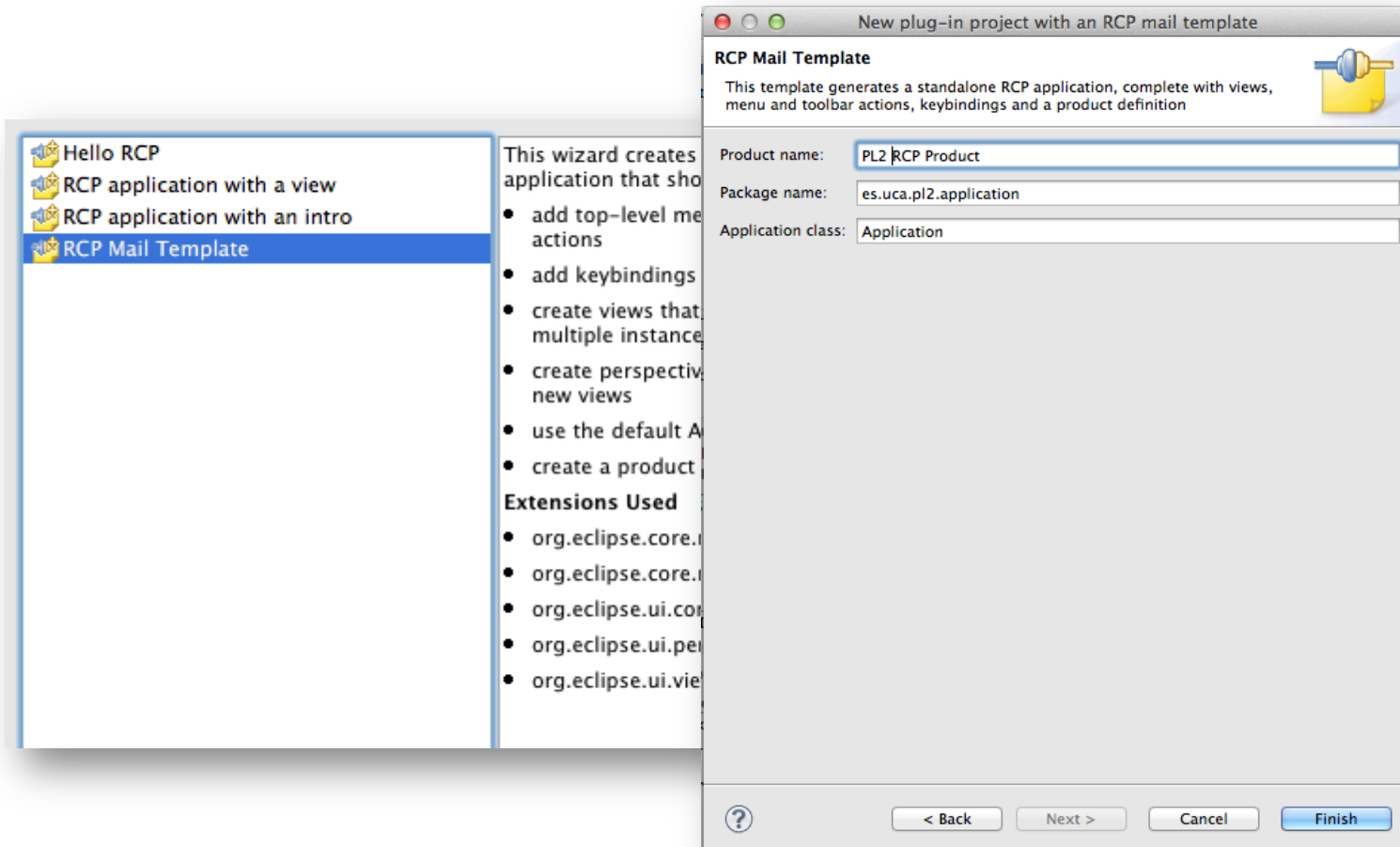
Enable API analysis

Rich Client Application

Would you like to create a rich client application? Yes No

< Back Next > Cancel Finish

Creación de una aplicación RCP (II)



Creación de una aplicación (III)

```
Application.java
package es.uca.pl2.application;

import org.eclipse.equinox.app.IApplication;

/**
 * This class controls all aspects of the application's execution
 */
public class Application implements IApplication {

    /* (non-Javadoc)
     * @see org.eclipse.equinox.app.IApplication#start(org.eclipse.swt.widgets.Display, org.eclipse.equinox.app.IApplicationContext)
     */
    public Object start(Display display, IApplicationContext context) {
        try {
            int returnCode = createPartControl(display);
            if (returnCode != 0) {
                return returnCode;
            }
            return IApplication.EXIT_OK;
        } finally {
            display.dispose();
        }
    }

    /* (non-Javadoc)
     * @see org.eclipse.equinox.app.IApplication#stop()
     */
    public void stop() {
        if (!Platform.isAndroid()) {
            return;
        }
    }
}

View.java
private Text messageText;

public void createPartControl(Composite parent) {
    Composite top = new Composite(parent, SWT.NONE);
    GridLayout layout = new GridLayout();
    layout.marginHeight = 0;
    layout.marginWidth = 0;
    top.setLayout(layout);
    // top banner
    Composite banner = new Composite(top, SWT.NONE);
    banner.setLayoutData(new GridData(GridData.HORIZONTAL_ALIGN_FILL, GridData.VERTICAL_ALIGN_FILL));
    layout = new GridLayout();
    layout.marginHeight = 5;
    layout.marginWidth = 10;
    layout.numColumns = 2;
    banner.setLayout(layout);

    // setup bold font
    Font boldFont = JFaceResources.getFontRegistry().getBold(JFaceResources.DEFAULT_FONT);

    Label l = new Label(banner, SWT.WRAP);
    l.setText("Subject:");
    l.setFont(boldFont);
    l = new Label(banner, SWT.WRAP);
    l.setText("This is a message about the cool Eclipse RCP!");

    l = new Label(banner, SWT.WRAP);
    l.setText("From:");
    l.setFont(boldFont);
}
```

Test de una aplicación RCP

es.uca.pl2.plugin es.uca.pl2.applicatio Application.java Activator.java View.java

Overview

General Information

This section describes general information about this plug-in.

ID:

Version:

Name:

Vendor:

Platform Filter:

Activator:

Activate this plug-in when one of its classes is loaded

This plug-in is a singleton

Plug-in Content

The content of the plug-in is made up of two sections:

- [Dependencies](#): lists all the plug-ins required on this plug-in's classpath to compile and run.
- [Runtime](#): lists the libraries that make up this plug-in's runtime.

Extension / Extension Point Content

This plug-in may define extensions and extension points:

- [Extensions](#): declares contributions this plug-in makes to the platform.
- [Extension Points](#): declares new function points this plug-in adds to the platform.

Testing

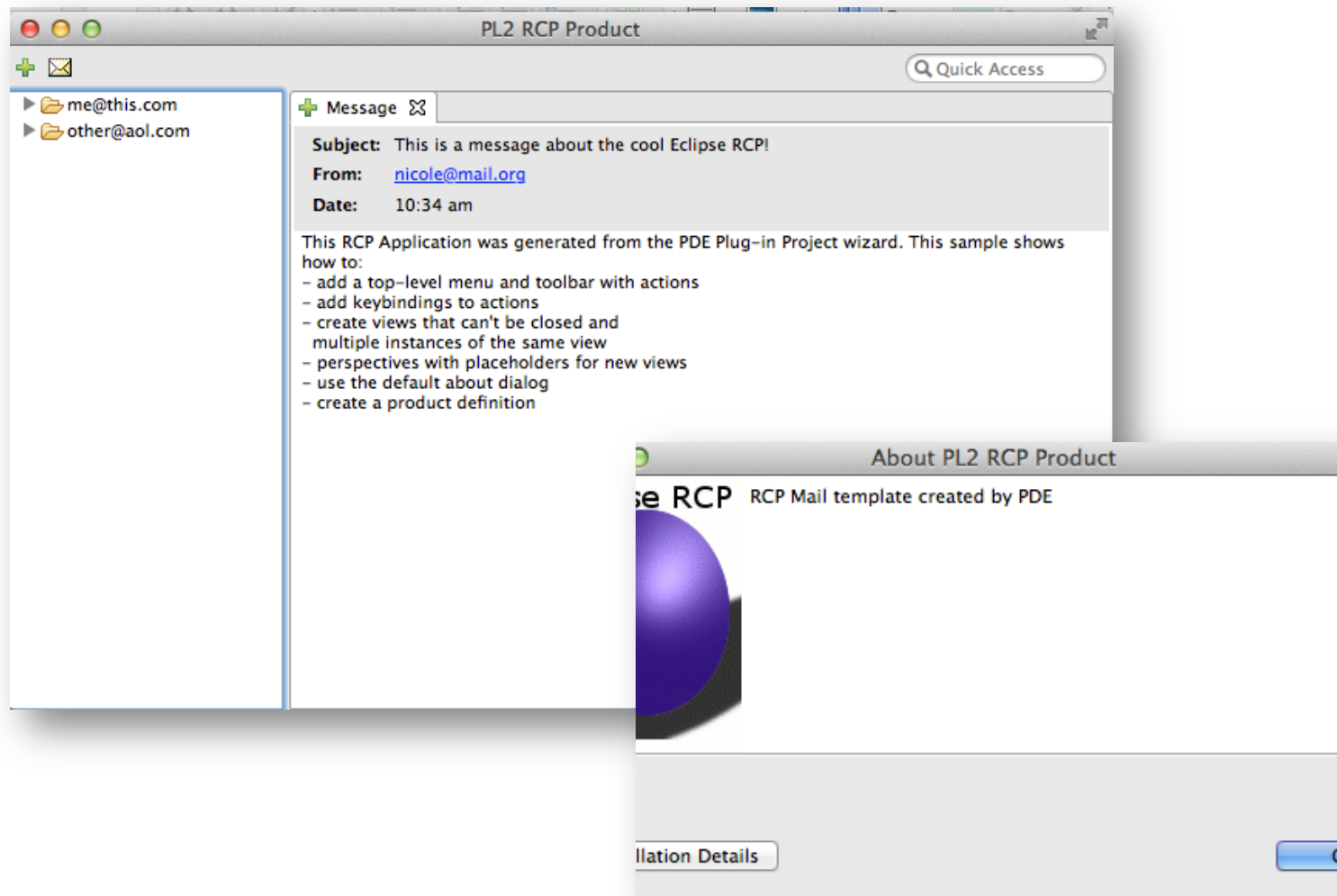
Test this plug-in by launching a separate Eclipse application:

-
-

Exporting

Overview | Dependencies | Runtime | Extensions | Extension Points | Build | MANIFEST.MF | plugin.xml | build.properties

Test de una aplicación RCP (II)





DESARROLLO DE EDITORES CON ECLIPSE



RESUMEN

¿Qué hemos aprendido hoy?

- Conocer Eclipse y sus proyectos principales
- Componentes principales del IDE
- Arquitectura modular basado en un kernel y un conjunto de plugins.
- Desarrollar plugins dentro del propio IDE o como aplicaciones independientes a partir de unas plantillas.



Procesadores de Lenguajes 2

La plataforma Eclipse

Curso 2013-2014

Iván Ruiz Rube

ivan.ruiz@uca.es