# Ejercicio 4

*Creación de plugins para Eclipse*

**Inicio: 24 de Febrero**
**Entrega: 6 de Marzo a las 11:00 a.m.**

El primer apartado del ejercicio propone extender Eclipse con un botón que muestre un cuadro de diálogo con un consejo aleatorio. Esta parte del ejercicio será guiada, a modo de tutorial, para que el estudiante se familiarice con la creación de plugins en Eclipse. En el segundo apartado, que también será guiado, se pide que las fuentes de datos con los consejos puedan añadirse dinámicamente. Finalmente, en el tercer apartado, el estudiante deberá extender de manera autónoma el plugin desarrollado en los apartados anteriores.

---

## Apartado 1: El consejo del día (3 puntos)

Se pide extender Eclipse con un botón que muestre un consejo aleatorio, de entre una lista de consejos almacenada en un fichero de texto. Para ello se utilizarán algunos puntos de extensión que permiten añadir una nueva opción de menú a Eclipse, la cual mostrará un cuadro de diálogo con el consejo. Posteriormente, modificaremos el plugin para que sea extensible, permitiendo añadir nuevas fuentes de consejos al mismo.

### Paso 1. Crear proyecto de tipo plugin

Un proyecto de tipo plugin es similar a un proyecto Java, pero además incluye editores para configurar el plugin. La Figura 1 muestra el asistente para crear un plugin, que es accesible mediante el menú `File → New → Project → Plug-in Project` de Eclipse. La primera página permite configurar el nombre del proyecto y otros aspectos como el *source folder*. Lo habitual es mantener las opciones por defecto. En la segunda página es importante poner un ID único al plugin, así como indicar si el plugin hará uso de la interfaz gráfica (en este caso sí, ya que queremos añadir un botón). Hay una tercera página que permite seleccionar una plantilla para crear un plugin con algún comportamiento predefinido. En este caso no usaremos ninguna, sino que crearemos el plugin desde cero.

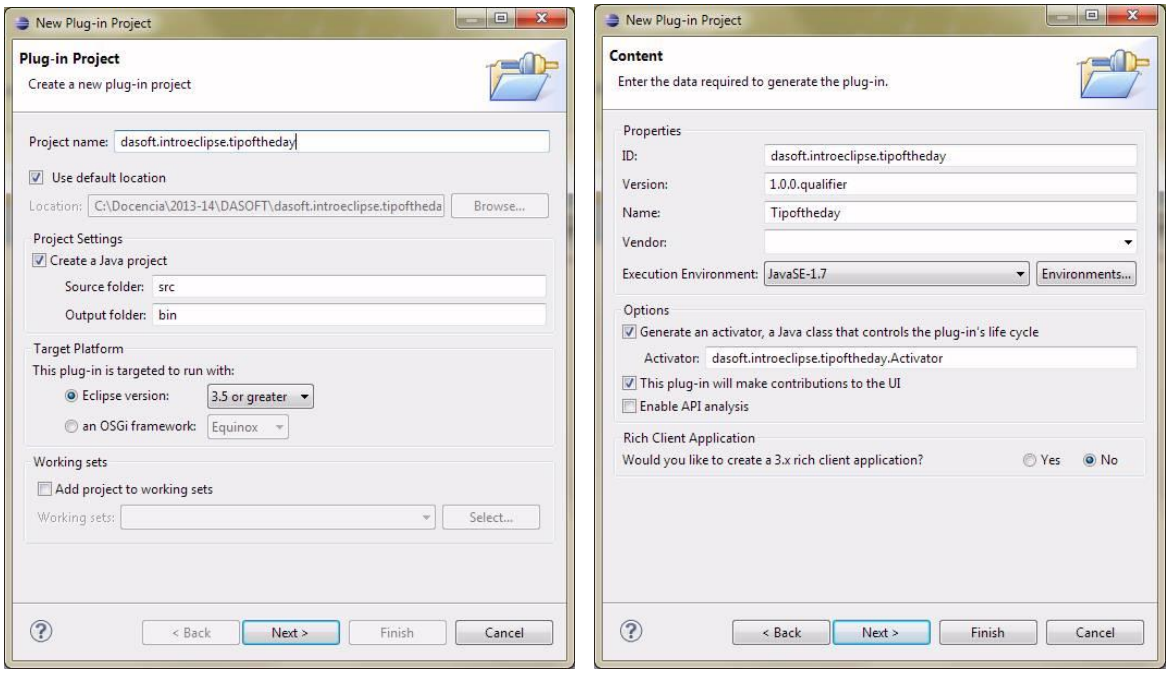


*Figura 1: Asistente de creación de un proyecto de tipo plugin*

Una vez introducida esta información, pulsamos `Finish`. Se creará un proyecto como el que muestra la Figura 2. El fichero `MANIFEST.MF` (abierto inicialmente) contiene la información de configuración del plugin. En el árbol de la

izquierda del *workbench*, el elemento *Plug-in Dependencies* hace referencia a las librerías que se añaden al *classpath* del proyecto debido a las dependencias del plugin.

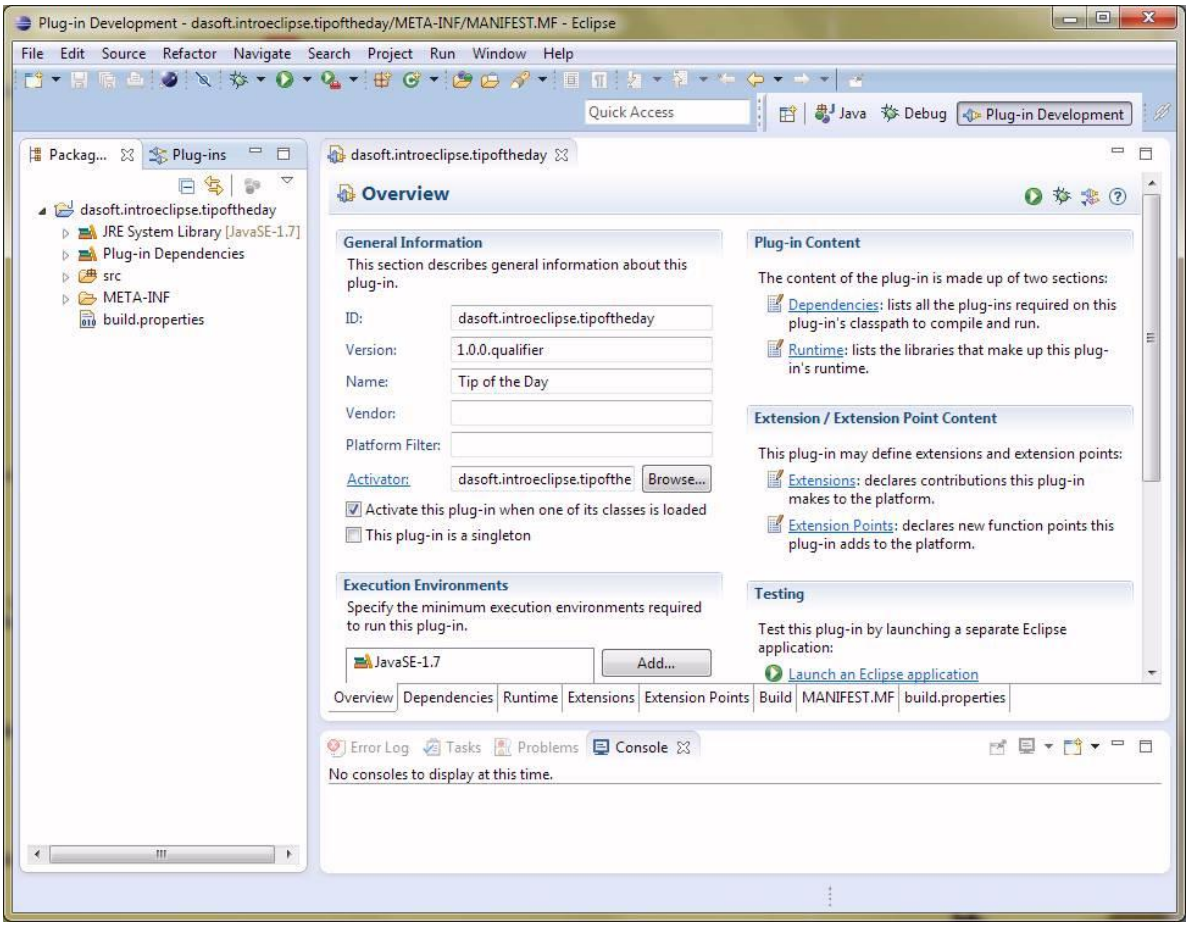


*Figura 2: Fichero MANIFEST.MF de configuración de un plugin*

**Paso 2. Instanciar punto de extensión**

Para extender Eclipse (por ejemplo con un botón) debemos instanciar alguno de los puntos de extensión que ofrece. Para ello, ir a la pestaña `Extensions` del fichero `MANIFEST.MF`. A continuación pulsar `Add`, que mostrará la lista de todos los puntos de extensión disponibles. Para este ejercicio (crear un botón), seleccionaremos el punto de extensión `org.eclipse.ui.commands`. Algunos puntos de extensión disponen de plantillas que completan la información que requiere el plugin con valores por defecto. En este caso, seleccionaremos la plantilla disponible en `Available templates for commands`, de tal modo que se instanciarán automáticamente otros puntos de extensión necesarios. Tras seleccionar la plantilla, iremos a la segunda ventana del asistente, que pide introducir el nombre de la clase que contendrá el código a ejecutar cuando se pulse el botón. Se recomienda dar un nombre apropiado a dicha clase (es decir, no dejar la que te propone por defecto).

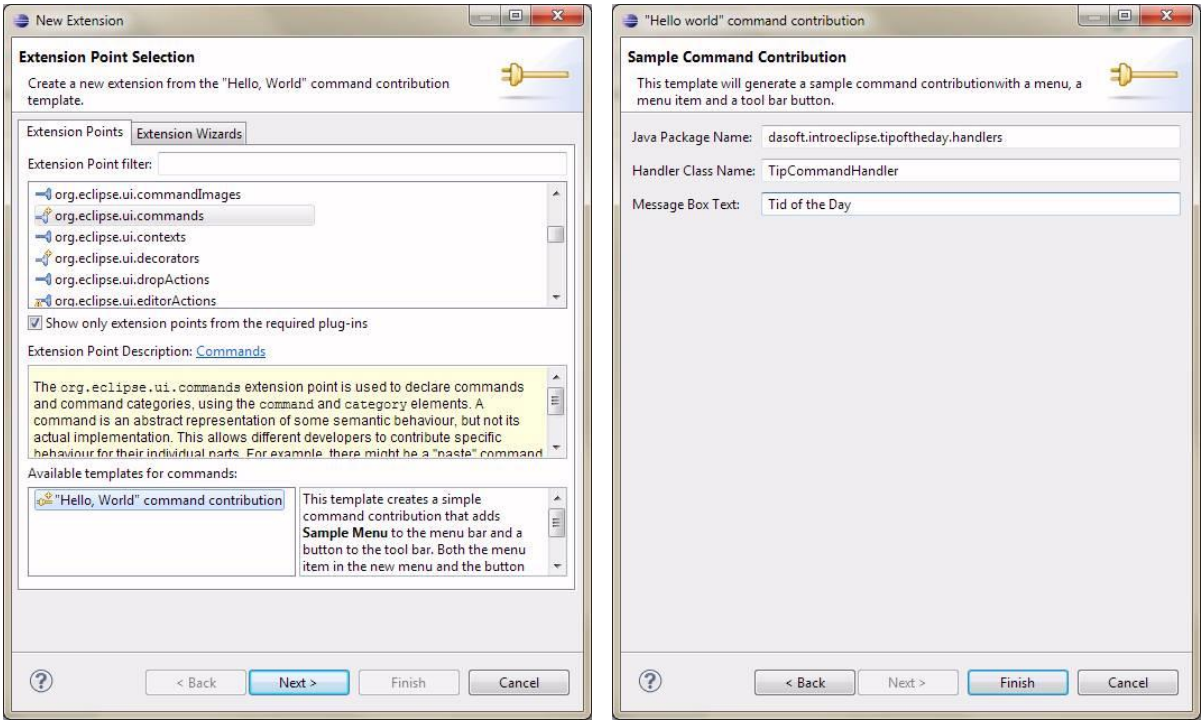


*Figura 3: Asistente para instanciar un punto de extensión*

La Figura 4 muestra los puntos de extensión que se instancian tras finalizar el asistente:

- `org.eclipse.ui.commands`: permite definir un comportamiento (un comando) y asociarle un id
- `org.eclipse.ui.handlers`: permite definir la clase que implementará el comportamiento de un comando
- `org.eclipse.ui.bindings`: permite definir una combinación de teclas, que activará un comando
- `org.eclipse.ui.menus`: permite definir menús, a través de los cuales se ejecutará un comando

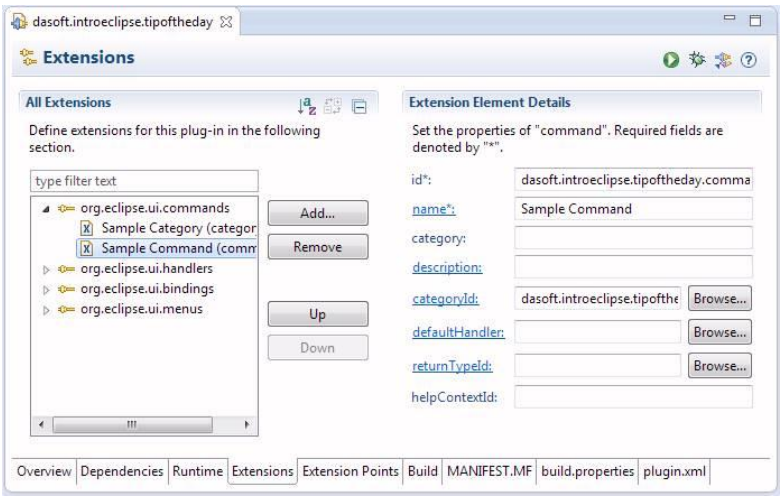


*Figura 4: Instancia de punto de extensión*

La instanciación de los puntos de extensión se realiza en el fichero `plugin.xml`, el cual se crea automáticamente al comenzar a usar puntos de extensión. El editor en forma de árbol mostrado en la Figura 4 es simplemente una forma más cómoda de editar este fichero. Puedes acceder al contenido real del fichero yendo a la pestaña `plugin.xml` del fichero `MANIFEST.MF`. Por ejemplo, el siguiente listado muestra cómo se instancia el punto de extensión `org.eclipse.ui.commands`. Este punto de extensión permite crear categorías (etiqueta `category`) e indicar que un comando (etiqueta `command`) pertenece a cierta categoría. En el listado, se ha modificado a mano el valor de los atributos `name` generados automáticamente para la categoría y el comando, para poner otros más apropiados. Por la misma razón, también se ha modificado el atributo `id` asignado por defecto al comando. Se pueden añadir tantos comandos y categorías como se necesite, definiendo nuevas entradas en el fichero siguiendo el mismo formato.

```
1   <extension point="org.eclipse.ui.commands">
2      <category
3            id="dasoft.introeclipse.tipoftheday.commands.category"
4            name="Tip of the day">
5      </category>
6      <command
7            categoryId="dasoft.introeclipse.tipoftheday.commands.category"
8            id="dasoft.introeclipse.tipoftheday.commands.tipOfTheDay"
9            name="Tip of the day">
10     </command>
11   </extension>
12  <extension point="org.eclipse.ui.handlers">
13     <handler
14           class="dasoft.introeclipse.tipoftheday.handlers.TipCommandHandler"
15           commandId="dasoft.introeclipse.tipoftheday.commands.tipOfTheDay">
16     </handler>
17  </extension>
```

Los elementos del punto de extensión suelen tener un atributo `id` que permite referenciarlos desde otros elementos. Por ejemplo, al instanciar el punto de extensión `org.eclipse.ui.handlers` (líneas 12-17), el atributo `commandId` se refiere al comando creado con el punto de extensión `org.eclipse.ui.commands` mediante su `id`. Por otra parte, el atributo `class` indica la clase que contendrá el código que se ejecutará al invocar el comando. En concreto, se ejecutará el contenido del método `execute` de la clase. Será el plugin anfitrión (es decir, el plugin que definió el punto de extensión, en este caso Eclipse) quien se encargará de instanciar la clase e invocar a los métodos adecuados cuando sea necesario.

```
1   public class TipCommandHandler extends AbstractHandler {
2      public Object execute(ExecutionEvent event) throws ExecutionException {
3         IWorkbenchWindow window = HandlerUtil.getActiveWorkbenchWindowChecked(event);
4         MessageDialog.openInformation(
5               window.getShell(),
6               "Tip of the day",
7               "A quien madruga dios le ayuda");
8         return null;
9      }
10  }
```

En el fichero `plugin.xml` también se ha instanciado el punto de extensión `org.eclipse.ui.menus`, que permite extender Eclipse con nuevas entradas en el menú principal (líneas 2-13) y botones en la barra de herramientas (líneas 14-23). Estas extensiones hacen referencia al comando creado con `org.eclipse.ui.commands`, Además, es posible configurar, entre otras cosas, el texto que se mostrará en el menú o el icono en el caso del botón.

```
1   <extension point="org.eclipse.ui.menus">
2      <menuContribution locationURI="menu:org.eclipse.ui.main.menu?after=additions">
3         <menu
4              id="dasoft.introeclipse.tipoftheday.menus.sampleMenu"
5              label="Tips"
6              mnemonic="M">
7         <command
8              commandId="dasoft.introeclipse.tipoftheday.commands.tipOfTheDay"
9              id="dasoft.introeclipse.tipoftheday.menus.tipOfTheDay"
10             mnemonic="S">
11        </command>
12     </menu>
13    </menuContribution>
14    <menuContribution locationURI="toolbar:org.eclipse.ui.main.toolbar?after=additions">
15       <toolbar id="dasoft.introeclipse.tipoftheday.toolbars.sampleToolbar">
16          <command
17               commandId="dasoft.introeclipse.tipoftheday.commands.tipOfTheDay"
18                icon="icons/sample.gif"
19                id="dasoft.introeclipse.tipoftheday.toolbars.tipOfTheDay"
20                tooltip="Tip of the day">
21          </command>
22       </toolbar>
23    </menuContribution>
24  </extension>
```

**Paso 3. Dependencias con otros plugins**

Un plugin puede definir puntos de extensión, y en muchas ocasiones permiten que se haga uso de ellos exponiendo un API. Para que un plugin pueda acceder a la funcionalidad publicada por otro plugin, debe indicar que hay una dependencia con él. Las dependencias se definen en la pestaña `Dependencies` del fichero `MANIFEST.MF`, en la lista `Required Plug-ins`. En este ejemplo, verás que por defecto se depende de los plugins `org.eclipse.ui` y `org.eclipse.core.runtime`. Si se necesitara hacer uso de clases o puntos de extensión definidos en otros plugins, habría que añadirlos a esta lista de dependencias.

**Paso 4. Exportar paquetes**

Las clases definidas en un plugin sólo son accesibles desde otros plugins si están en la lista de paquetes exportados del plugin. Esto es necesario si se está construyendo una extensión de Eclipse que consta de varios plugins que comparten clases entre sí. Si necesitas hacer visibles clases definidas en un plugin para poder usarlas en otro plugin, debes añadir los paquetes de las clases a la lista `Exported Packages` que se encuentra en la pestaña `Runtime` del fichero `MANIFEST.MF`. En este ejemplo no es necesario.

**Paso 5. Leer recursos desde un plugin**

A veces, los plugins contienen recursos (imágenes, ficheros de texto, etc.) que hay que procesar programáticamente. Para ello hay que tener en cuenta que los plugins normalmente se despliegan comprimidos en ficheros `.jar`, por lo que no es posible apuntar directamente al recurso para leerlo como si fuera un fichero del sistema de ficheros. En su lugar, hay que usar el API de Eclipse para obtener una URL que apunte al recurso y obtener un `InputStream` para procesarlo.

En el ejemplo, supondremos que la lista de consejos a mostrar está en el fichero `consejos.txt`, de tal forma que hay que leerlo para seleccionar aleatoriamente el que se mostrará al pulsar el botón. El siguiente listado muestra una estrategia para leer el fichero. Las líneas 2, 3 y 6 son particularmente importantes, ya que se encargan de obtener un *stream* de lectura sin decir explícitamente la ruta completa del sistema de archivos en que está el fichero. En la línea 2, el parámetro que se le pasa a `Platform.getBundle` es el `ID` que hayamos definido para el plugin (ver Figura 2). En la línea 3, se debe indicar el *path* del fichero a leer, relativo al proyecto actual.

```
1   List<String> consejos = new ArrayList<String>();
2   Bundle bundle = Platform.getBundle("dasoft.introeclipse.tipoftheday");
3   URL fileURL = bundle.getEntry("data/consejos.txt");
4
5   try {
6      InputStream    is = fileURL.openStream();
7      BufferedReader in = new BufferedReader(new InputStreamReader(is));
8      String consejo;
9      while ((consejo = in.readLine()) != null) consejos.add(consejo);
10     in.close();
11     is.close();
12  }
13  catch (IOException e) { e.printStackTrace(); }
```

Se deja como ejercicio al estudiante el modificar el método `execute` de la clase "handler" para que, una vez leído el fichero con la lista de consejos, seleccione uno de manera aleatoria, y lo muestre en el cuadro de diálogo.

**Paso 6. Probar y depurar un plugin**

Un plugin no puede probarse en la misma instancia de Eclipse en la que se está desarrollando. Una opción sería empaquetar el plugin e instalarlo en una nueva instancia de Eclipse, pero eso es costoso y sólo se hace cuando el plugin está estable. En su lugar, Eclipse ofrece un *launcher* especializado en ejecutar plugins, denominado *Eclipse Application*. Este *launcher* lanza una segunda instancia de Eclipse, igual que en la que se está desarrollando, pero que tiene desplegados los plugins que se están desarrollando. En esta segunda instancia, la carpeta que corresponde al *workspace* es diferente de la que se usa en tiempo de desarrollo.

Para probar el plugin que estamos desarrollando, seleccionar `Run → Run Configurations…`, y hacer doble click sobre `Eclipse Application`. De ese modo estamos creando un nuevo *launcher* que, al ejecutarse, lanzará la segunda instancia de Eclipse. En esta segunda instancia, debe haber un nuevo botón en la barra de herramientas que

mostrará el consejo del día al pulsarlo, así como un nuevo menú en la parte superior que también permite lanzar el consejo del día.

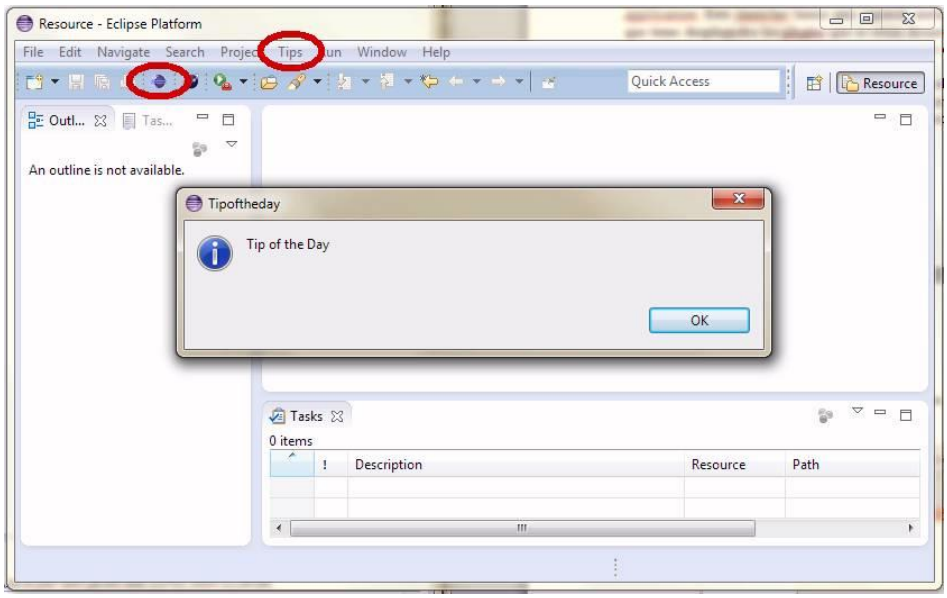


*Figura 5: Probando el plugin con el launcher de Eclipse*

Para depurar el plugin puedes utilizar el depurador normal de Eclipse, lanzando la segunda instancia con el lanzador de depuración. De este modo se pueden definir puntos de ruptura, inspeccionar valores y ejecutar paso-a-paso.

**Paso 7. Empaquetar y desplegar un plugin**

Una vez terminado el desarrollo de un plugin, hay que empaquetarlo para que pueda ser usado (desplegado) por otros usuarios en sus entornos Eclipse. Para ello, Eclipse hace uso de la idea de *feature*, que sirve para agrupar uno o más plugins relacionados en una unidad de instalación.

Para definir una *feature*, crear un proyecto de tipo *Feature* mediante el menú `File → New → Project → Plug-in Development → Feature Project`. Dentro del proyecto se creará el fichero `feature.xml` para configurar la *feature*. Si abres el fichero verás que contiene varias pestañas, cada una con distintas opciones de configuración. Las principales son `Overview` y `Plug-ins`, esta última para indicar qué plugins componen la *feature*. Selecciona la pestaña `Plug-ins` y añade a la lista `Plug-ins and Fragments` el plugin que hemos creado. Para ello haz click en el botón `Add`: aparecerá una ventana de búsqueda en la que tendremos que introducir el ID del plugin.

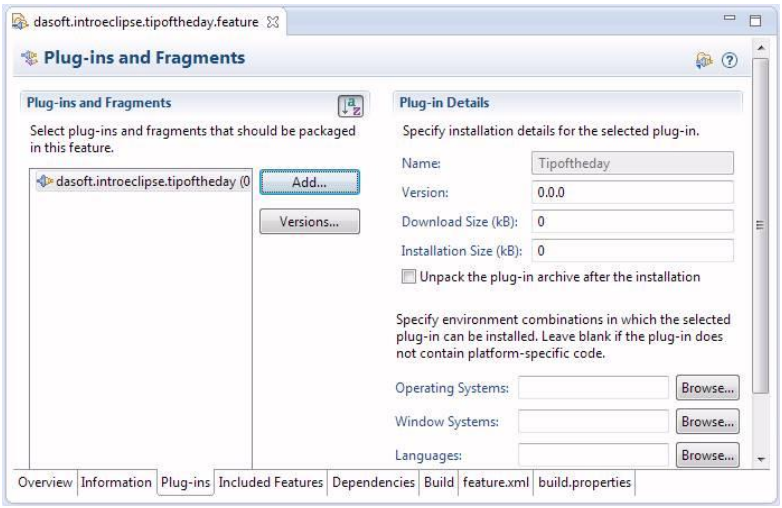


*Figura 6: Lista de plugins que contiene una feature*

A la hora de configurar el empaquetado de un plugin hay que tener en cuenta que el hecho de que éste funcione al probarlo en el entorno de desarrollo no garantiza que funcionará al instalarlo en una nueva instancia de Eclipse. La razón es que, en tiempo de desarrollo, todos los recursos y librerías están disponibles en el *workspace*, por lo que al empaquetar para el despliegue, es importante asegurarse de que la versión instalable también los contendrá. Así, se recomienda comprobar lo siguiente:

- el plugin empaquetado contiene todos los recursos necesarios. Para ello, el fichero `build.properties` de los plugins debe indicar los recursos (ej. ficheros de texto, imágenes, etc.) que deben incluirse en la distribución. En nuestro ejemplo, debemos añadir el fichero de consejos. Para ello, abre el fichero `build.properties` del plugin, y selecciona la carpeta que contiene este fichero en la lista `Binary Build`.
- el plugin empaquetado contiene todas las librerías externas necesarias. Para ello, el fichero `plugin.xml` de los plugins deben indicar si se utiliza alguna librería externa. En este ejemplo no se ha utilizado ninguna, así que no hace falta indicar nada. Si fuese necesario, estas librerías deberían indicarse en la pestaña `Runtime` del fichero.

Una vez configurada la *featur*e, sólo queda exportarla. La forma más sencilla es generar un fichero `.zip`. Para ello, abre el fichero `feature.xml`, selecciona la pestaña `Overview`, y ejecuta las etapas de exportación que aparecen en la sección `Exporting`:

- Sincronizar las versiones de los plugins con la versión de la *feature* (opción `Synchronize`). Aunque existen diversas opciones de sincronización, se recomienda hacer que la versión de la *feature* se copie en los plugins (opción por defecto). De este modo se evitan problemas al reinstalar diferentes versiones de una *feature*.
- Acceder al asistente de exportación (opción `Export Wizard`). Selecciona la opción `Archive File`, e indica el nombre del fichero `.zip` que contendrá el plugin empaquetado, listo para su distribución.

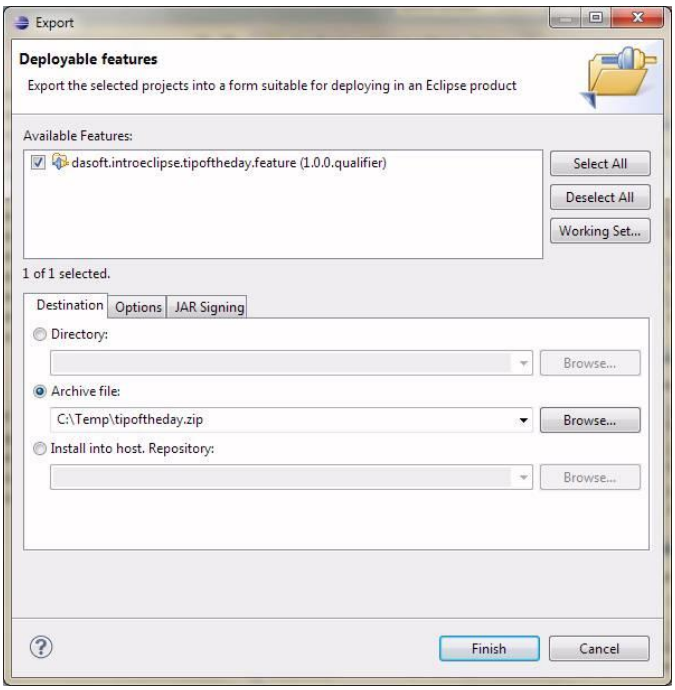


*Figura 7: Asistente para exportar el plugin empaquetado para su distribución*

Para comprobar si has generado correctamente la distribución del plugin, instálalo en Eclipse (nota: deselecciona el check-button *Group items by category* en el asistente de instalación, para que te aparezca la *feature* correspondiente a tu plugin). El botón y la opción de menú para mostrar el consejo del mes deberían aparecer tras instalar el plugin.

---

## Apartado 2: Añadiendo fuentes de datos dinámicamente (3 puntos)

Un punto de extensión permite a un plugin indicar que es posible extender su funcionalidad base con otra funcionalidad proporcionada por otros plugins. Un punto de extensión se define mediante un esquema XML que especifica qué datos deben proporcionarse para poder instanciarlo, y así proveer esa funcionalidad adicional.

En el ejemplo, se está interesado en permitir que otros plugins proporcionan nuevas fuentes de consejos. El plugin anfitrión (que acabamos de desarrollar) ya proporciona algunos consejos, pero ahora queremos que añada a su lista de consejos los que definan aquellas fuentes añadidas dinámicamente a través del punto de extensión.

Una forma de añadir una fuente de consejos es crear una interfaz, por ejemplo `ITipSource`, de manera que el punto de extensión creado indique que toda extensión del plugin debe proporcionar una clase que implemente esta interfaz. La interfaz debe incluir un método, por ejemplo `getTips`, que devuelva la lista de consejos proporcionada por la nueva fuente.

La creación del esquema XML para definir el punto de extensión se puede realizar en la pestaña `Extensions Points` del fichero `plugin.xml`, tal como muestra la Figura 8.

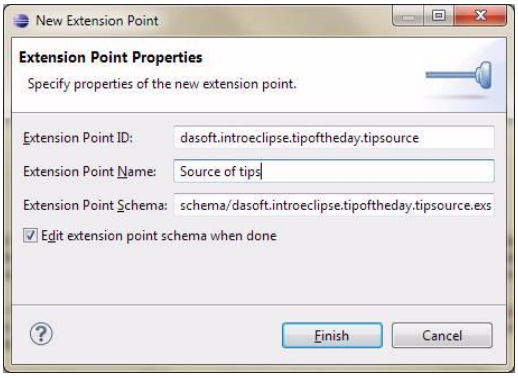


*Figura 8: Asistente de creación de un punto de extensión*

Al crear el punto de extensión, se añadirá automáticamente un fichero `.exsd` a la carpeta `schema` del proyecto. Al seleccionar la pestaña `Definition` de este fichero, se mostrará un editor en forma de árbol que permite definir el esquema del punto de extensión cómodamente. Se pueden definir tres tipos de elementos:

- elementos XML
- secuencias de selección (`sequence` o `choice`) que referencian a un elemento XML
- atributos

La Figura 9 muestra la definición del esquema para el punto de extensión del ejercicio. Dicho esquema (que deberás construir) incluye el elemento XML `TipSource`, que tiene un atributo llamado `class` en el que se indicará la clase que implementará la fuente de consejos. Este atributo es de tipo `java`, lo que indica que el valor que se le asigne debe ser una clase Java. Además, se ha indicado que la clase que se proporcione al instanciar el punto de extensión debe implementar la interfaz `ITipSource`. Finalmente, se ha añadido un elemento de tipo `Sequence` a la definición del punto de extensión, lo que permitirá que sus instancias puedan definir uno o más elementos de tipo `TipSource`.

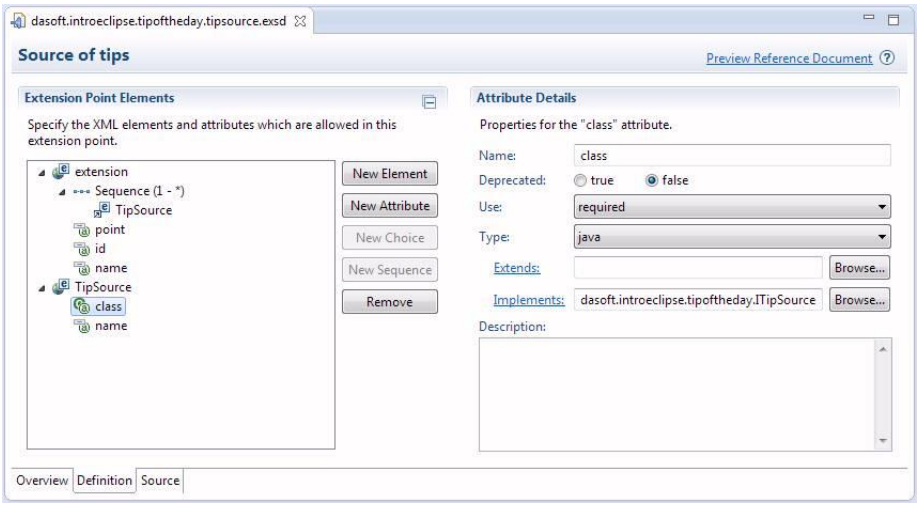


*Figura 9: Configuración de un punto de extensión*

La definición del esquema XML permitirá al plugin extensor configurar la información requerida por el plugin anfitrión, el cual debe leer la información proporcionada y configurarse en consecuencia. Para ello, Eclipse ofrece un API que permite consultar qué puntos de extensión han sido instanciados y leer esa información.

El siguiente listado muestra cómo usar este API El método `Platform.getExtensionRegistry` permite acceder al registro de los puntos de extensión (línea 1) y obtener la lista de instancias de un punto de extensión dado (líneas 2-3). La interfaz `IConfigurationElement` representa de forma genérica los nodos XML del punto de extensión, y proporciona métodos para preguntar por los hijos del punto de extensión y el valor dado a sus atributos. En este caso se ha usado el método `createExecutableExtension` (línea 8), que instancia una clase especificada en un atributo de tipo `java`.

```
1   IExtensionRegistry      registro    = Platform.getExtensionRegistry();
2   IConfigurationElement[] extensiones =
3     registro.getConfigurationElementsFor("dasoft.introeclipse.tipoftheday.tipsource");
4
5   for (IConfigurationElement e : extensiones) {
6      if (e.getName().equals("TipSource")) {
7         try {
8            ITipSource fuenteDatos = (ITipSource) e.createExecutableExtension("class");
9            for (String consejo : fuenteDatos.getTips()) {
10              consejos.add(consejo);
11           }
12        }
12        catch (CoreException e1) {
13           Status estado = new Status(Status.ERROR, Activator.PLUGIN_ID, "Expected ITipSource");
15           ILog   logger = Activator.getDefault().getLog();
16           logger.log(estado);
17        }
18     }
19   }
```

Usando como guía el listado anterior, modifica la clase "handler" del plugin para que, en su constructor, lea todas las instancias del punto de extensión definido para cargar las fuentes de datos que hayan declarado otros plugins. Recuerda crear la interfaz `ITipSource` con un método (`getTips`) que deberán implementar las clases que se usen en las instancias del punto de extensión. También debes recordar añadir el paquete que contiene esa interfaz a la lista de paquetes exportados del plugin, para que sea accesible desde otros plugins. Esto se define en la pestaña `Runtime` del fichero `plugin.xml`.

Puedes instanciar el nuevo punto de extensión creado igual que cualquier otro punto de extensión: crea un proyecto de tipo plugin, añade el plugin original a sus dependencias, e instancia el punto de extensión (puedes recordar cómo se realiza esto mirando los pasos anteriores del ejercicio). El fichero `plugin.xml` del nuevo plugin debería contener algo similar a la siguiente descripción, que correspondería a la instancia del punto de extensión creado:

```
1   <extension
2      point="dasoft.introeclipse.tipoftheday.tipsource">
3      <TipSource
4          name="Eclipse tips"
5          class="dasoft.introeclipse.fuentedeconsejos.EclipseTips" />
6   </extension>
```

---

## Apartado 3: Consejos multi-idioma (4 puntos)

Se pide extender el plugin desarrollado en los apartados anteriores para que permita mostrar consejos en el idioma de preferencia del usuario. A continuación se detallan los pasos que se pide realizar:

- Define un nuevo punto de extensión que permita definir una lista de idiomas (ej. español, inglés, etc.).
- Modifica el punto de extensión desarrollado en el apartado 2 para que, al añadir una nueva fuente de consejos, se tenga que indicar el idioma en el que están definidos. Si al cargar una fuente de consejos se detecta que el idioma no se ha definido mediante un punto de extensión, la fuente no se cargará.
- Añade al menú general creado en el apartado 1 un submenú que permita seleccionar el idioma de los consejos. Al seleccionar este submenú, se mostrará un cuadro de diálogo con la lista de los idiomas definidos mediante

puntos de extensión, y se permitirá seleccionar uno. De este modo, cuando se quiera mostrar un consejo, deberá elegirse aleatoriamente de entre las fuentes definidas en ese idioma. La selección de idioma no podrá hacerse mediante un botón, sólo mediante esta opción de menú.
Utiliza SWT para crear el cuadro diálogo. En este caso, puedes usar `ElementListSelectionDialog`, que muestra un cuadro de diálogo con una lista de elementos seleccionables. Si quieres aprender más, en la siguiente URL tienes ejemplos de uso de SWT: http://www.vogella.com/tutorials/EclipseDialogs/article.html.

- Actualiza el fichero `.zip` con el plugin empaquetado.

## Normas de Entrega

Se pide entregar:

- ✓ el proyecto/s con la definición del plugin
- ✓ el proyecto/s con la definición de la *feature*
- ✓ el fichero `.zip` con el plugin empaquetado
- ✓ un plugin de prueba donde se demuestre la definición de idiomas (si se ha implementado el tercer apartado) y de fuentes de consejos

Todo se comprimirá en un único fichero zip/rar, que se entregará a través de moodle antes de la fecha de entrega indicada en la cabecera del enunciado.

El nombre del fichero deberá incluir el nombre y apellido del estudiante.