Tutorial para escribir un pequeño plugin para jEdit

Jairo Martínez

Tutorial para escribir un pequeño plugin para jEdit

por Jairo Martínez

Historial de revisiones

Revisión 1.0 1 Junio 2002 Revised by: jm

Revisión 1.1 10 de Junio Revised by: jid

Revisión 1.2 12 de Junio Revised by: jm

Tabla de contenidos

Capítulo 1. Aspectos básicos de Jedit

1.1. ¿Qué es Jedit?

jEdit es un editor de texto orientado hacia programadores y escrito en Java. Fue desarrollado por Slava Pestov y otros.

Sus características principales son :

- Fácil de usar, extremadamente configurable y posee un amplio conjunto de opciones, dentro de las que se encuentra el coloreado de sintaxis para 70 lenguajes, tabulación automática, ilimitado deshacer/rehacer, entre muchas otras.

- Utiliza BeanShell, como lenguaje de *scripting* que permite el desarrollo de macros para jEdit. Pocos macros son incluidos con jEdit; sin embargo se pueden encontrar muchos más en la dirección de Internet *http://community.jedit.org* (http://community.jedit.org)

- Soporta plugins. Mas de 50 plugins estan actualmente disponibles para automatizar un amplio rango de tareas. Los plugins pueden ser descargados de Internet y instalados mediante el *"plugin manager"*, el cual viene incluido con jEdit. Para encontrar más información sobre los *plugins* de jEdit puede consultar en esta dirección de internet *http://plugins.jedit.org* (http://plugins.jedit.org)

Las siguientes son algunas vistas de pantalla de jEdit corriendo sobre diferentes plataformas:

Windows

jEdit corriendo sobre Windows
Linux

jEdit corriendo sobre Linux

1.2. Requisitos mínimos

jEdit requiere Java 2 (o Java 1.1 con Swing 1.1). La versión de Java recomendada para utilizar jEdit es Java 2 versión 1.3.

1.3. Licencia

jEdit es software libre, por lo que se da la libertad de distribuirlo bajo los términos de la Licencia Publica General GNU (la versión 2 o superiores).

Con respecto a los plugins disponibles para jEdit, a menos que esté especificado de manera diferente en la documentación, estos se encuentran licenciados para uso y distribución bajo los términos de la Licencia Publica General GNU (versión 2 o superiores).

1.4. Descargar el programa desde Internet y obtener información adicional

Para descargar el programa desde internet y encontrar información detallada sobre los diferentes aspectos del programa (incluyendo su código fuente) puede dirigirse a la siguiente dirección de internet: *http://www.jedit.org* (http://www.jedit.org).

1.5. Explicación de la arquitectura de jEdit

jEdit es una aplicación monousuario que trabaja de manera local. Sin embargo, cuenta con algunos servicios para ser utilizados sobre una conexión de red como lo son el descargar *plugins* y realizar transferencias de archivos mediante un *plugin* para manejar FTP.

Para desarrollar macros o plugins para jEdit es necesario conocer al API (*aplication programmer interface*) que se incluye con la aplicación. Esta está conformada por:

- Comandos de BeanShell
- Clases generales de jEdit

- Clases de EditBus

Los comandos de BeanShell sólo pueden ser usados por las macros. Las clases generales de jEdit son usadas tanto para el desarrollo de macros como para el de *plugins*. Las clases de EditBus cubren el manejo del sistema de mensajes EditBus, el cual es utilizado principalmente por los *plugins*.

Para consultar información detallada sobre el API de jEdit puede consultar los documentos de ayuda de jEdit, a los cuales puede acceder mediante el menú de ayuda de jEdit

Capítulo 2. Creación de un *plugin* básico

2.1. ¿Qué es un *plugin*?

Un *plugin* es un programa adicional que puede ser añadido a una aplicación para aumentar la funcionalidad de esta. Este programa adicional es cargado y corrido por la aplicación principal.

Al igual que jEdit, sus *plugin* son escritos primordialmente en Java. Para escribir en *plugin* aunque se requiere cierto conocimiento del lenguaje, no se necesita ser un experto en Java. Si usted puede escribir una aplicación útil en Java, usted puede escribir un *plugin*.

A continuación, desarrollaremos un pequeño *plugin* con el objetivo de ilustrar las aspectos básicos que debe tener un *plugin* desarrollado para jEdit.

Para escribir su primer *plugin*, usted necesitará:

- Java 2, Standard Edition. (se recomienda la versión 1.3)
- jEdit (ultima versión estable en la fecha de escritura de este documento : 4.0). Se puede descargar del sitio: *http://www..jedit.org* (http://www.jedit.org)

Tambien es recomendable utilizar alguna herramienta que permita desarrollar aplicaciones para Java de manera rápida y eficiente, como por ejemplo JBuilder. Sin embargo, esto no es un requisito. Se pueden desarrollar los diferentes archivos mediante el uso de un editor de texto como emacs, notepad, editplus o el mismo jEdit.

2.2. *Plugin* Hello World!

Desarrollaremos un *plugin* bastante sencillo, en donde se muestran las características mínimas que debe tener un *plugin* para jEdit. Este *plugin* tendrá como única función mostrar en un cuadro de dialogo la frase *"Hello World !"*

Los pasos que se seguiran para desarrollar el *plugin* son los siguientes:

1.

2. Escribir la clase principal del *plugin*, la cual llevará el nombre de `HelloWorldPlugin`

3. Escribir la clase encargada de la visualización del *plugin*, a la que llamaremos `HelloWorld`

4. Desarrollar los archivos donde se encuentran los recursos a utilizar por el *plugin*

5. Compilar los archivos que contienen el código fuente de las clases que se han desarrollado.

6. Empaquetar el *plugin* en un archivo JAR

7. Ubicar el archivo JAR en el directorio donde jEdit busca sus *plugins*

2.2.1. Escribiendo la clase principal del *plugin*

Empezaremos a desarrollar el *plugin*, escribiendo su clase principal. Esta clase la desarrollaremos en un archivo con el siguiente nombre: , escribiendo su clase principal. El código fuente de esta clase puede encontrarlo en:

En caso de que el *plugin* deba responder a cambios de estado de la aplicación principal, la clase principal del *plugin* debe heredar de la clase `EBPlugin`. Sin embargo, dado que en nuestro caso el *plugin* no necesita tanta complejidad, heredaremos de la clase `EditPlugin`.

Para poder heredar de esta clase hay que incluir las librerías de jEdit, por lo que nuestra primera línea de código será:

```
import org.gjt.sp.jedit.*;
```

Luego, escribiremos la siguiente parte del código en donde especificaremos que la clase `HelloWorldPlugin` hereda de la clase `EditPlugin` y que posee una única función encargada de agregar el item de menú de nuestro *plugin* al menú de plugins de jEdit. El resultado final del código escrito para la clase `HelloWorldPlugin` es el siguiente:

```
import org.gjt.sp.jedit.*;

import java.util.Vector;
```

```
public class HelloWorldPlugin extends EditPlugin
{
    public void createMenuItems(Vector menuItems)
    {
          menuItems.addElement(GUIUtilities.loadMenuItem("HelloWorld"));
     }
}
```

2.2.2. Escribiendo la clase `HelloWorld`

A continuación, desarrollaremos la clase que está encargada de la visualización del *plugin*. A esta clase le daremos por nombre `HelloWorld`. El código fuente de esta clase puede encontrarlo en:

Para implementar la visualización, esta clase heredará de la clase `JDialog` de Java. Además implementará dos métodos encargados de desplegar y cerrar el cuadro de diálogo.

Esta clase tendrá como atributo privado un objeto de la clase `View`, que proviene de la aplicación principal y es enviado hacia el *plugin* cuando se realiza el llamado de éste. Esto le da la posibilidad al *plugin* de interactuar con este objeto, el cual está encargado de contener la mayoría de componentes visibles del programa.

El siguiente es el código para esta clase:

```
import javax.swing.*;
import org.gjt.sp.jedit.*;

public class HelloWorld extends JDialog
{
// private members
private View view;

     public HelloWorld(View view)
    {
       super(view,jEdit.getProperty("HelloWorld.title"),false);

              this.view = view;

       JPanel content = new JPanel();
       setContentPane(content);

       JLabel caption = new JLabel(jEdit.getProperty("HelloWorld.caption"));
       content.add(caption);

       pack();
       GUIUtilities.loadGeometry(this, "HelloWorld");
       setDefaultCloseOperation(DISPOSE_ON_CLOSE);
       show();
```

```
    }


    public void dispose()
    {
       view = null;
       GUIUtilities.saveGeometry(this, "HelloWorld");
       super.dispose();
    }
}
```

2.2.3. Recursos requeridos por el *plugin*

2.2.3.1. Acciones

El catálogo de acciones de usuario del *plugin* es el recurso usado por el API de *plugins* para obtener los nombres y las definiciones de las acciones del usuario. A continuación, desarollaremos el archivo que contendrá las acciones del *plugin* HelloWorld. El contenido de este archivo puede encontrarlo en:

```
<?xml version="1.0"?>

<!DOCTYPE ACTIONS SYSTEM "actions.dtd">

<ACTIONS>
<ACTION NAME="HelloWorld">
<CODE>
new HelloWorld(view);
</CODE>
</ACTION>
</ACTIONS>
```

Dentro de este archivo se define la única acción que tiene el *plugin*, en donde se llama una nueva instancia de la clase `HelloWorld`, enviándosele por parámetro el objeto `view` de la aplicación principal.

El código que aparece dentro del tag <CODE> sigue las reglas de sintaxis del código de BeanShell. Si desee encontrar mayor información sobre la definición de `actions.xml` o sobre BeanShell puede consultar la Guía de Usuario de jEdit, que se encuentra dentro de la ayuda del programa.

2.2.3.2. Propiedades

Para configurar un *plugin* de manera externa, se fija cierta información relevante para el *plugin* como lo es el nombre del *plugin*, el texto de los items del menú, el texto desplegado en el *plugin*, el nombre del archivo de documentación, el autor, la versión, entre otros en un archivo de texto.

Esto también permite que jEdit tenga acceso a las diferentes propiedades del *plugin*, mediante la extracción de las propiedades del éste del archivo de texto.

Para nuestro *plugin* utilizaremos el archivo `HelloWorld.props` , que ilustra el uso de estas propiedades. Puede encontrar su contenido en:

```
# Plugin properties
plugin.HelloWorldPlugin.name=Hello World !
plugin.HelloWorldPlugin.author=Jairo Martinez
plugin.HelloWorldPlugin.version=1.0
plugin.HelloWorldPlugin.docs=HelloWorld.html

# Menu item label
HelloWorld.label=Hello World !

# HelloWorld window
HelloWorld.title=Hello World plugin
HelloWorld.caption=Hello World !
```

2.2.4. Compilando el *plugin*

A la hora de compilar hay que tener cuidado en que se haya incluido dentro de la variable del sistema, `CLASSPATH` la ruta de las librerias de jEdit. Las librerias de jEdit se pueden encontrar en el directorio de instalación del programa en un archivo jar llamado `jedit.jar`.

Compile los archivos `HelloWorld.java` y `HelloWorldPlugin.java`, para obtener los archivos `HelloWorld.class` y `HelloWorldPlugin.class`.

2.2.5. Empaquetar en un archivo JAR

Ahora empaquetaremos los archivos que conformaran el plugin en un archivo JAR, al que denominaremos `HelloWorld.jar`.

Estos son los archivos que empaquetaremos dentro del archivo JAR:

- `HelloWorld.class`
- `HelloWorldPlugin.class`

- `actions.xml`
- `HelloWorld.props`

Una forma fácil de crear este archivo puede ser poner los archivos citados anteriormente en un directorio y luego utilizar el siguiente comando en ese directorio:

```
jar cf HelloWorld.jar *
```

El archivo JAR terminado lo puede encontrar en este archivo: `HelloWorld.jar` (HelloWorld1/HelloWorld.jar)

2.2.6. Ubicar el archivo JAR

Para terminar, se debe ubicar el archivo `HelloWorld.jar` en la carpeta: *`[directorio de instalacion de JEdit]`*`/jars`

jedit tree

2.2.7. Resultado final

Para observar el *plugin*, solo debemos ejecutar jEdit y luego hacer clic en el menú Plugins Hello World!

menu-plugin

Ahora debe aparecer una ventana de este estilo:

HelloWorld window

¡Felicitaciones! ha terminado de desarrollar su primer *plugin* para jEdit.

> **Importante:** Hasta este punto se puede tener ya una buena idea de cómo se desarrolla un *plugin*, sin embargo, vale la pena especificar claramente cuales son los requerimientos básicos para un *plugin*. Estos son:

- Debe estar empaquetado en uno o más archivos JAR y contener un archivo de clase que termine con el prefijo Plugin y que herede de la clase abstracta `EditPlugin`. Si el plugin debe responder a cambios de estado en la aplicación jEdit, esta clase debe ser heredar de la clase `EBPPlugin`.

- El archivo JAR debe contener, por lo menos, un archivo de propiedades que tenga como extensión `.props`. Estas propiedades deben dar información sobre como el *plugin* está definido.

- El archivo JAR debe contener las acciones que podrá realizar el *plugin* para que estas sean desplegadas dentro de la barra de menú de jEdit. Estas serán especificadas en un archivo de nombre `actions.xml`.

Capítulo 3. Agregando funcionalidad al *plugin*

A continuación buscaremos aumentar la funcionalidad del *plugin Hello World !* dando la posibilidad de mostrarse al estilo de un *"wizard"* o "asistente". Esto significa que el usuario será guiado paso a paso en el desarrollo de el proceso para realizar la tarea que cumple el *plugin*, mediante el uso de ventanas consecutivas, sobre las que le usuario podrá avanzar o retroceder haciendo uso de los botones "Siguiente o "Anterior", respectivamente.

Anteriormente la función que realizaba el *plugin* era imprimir el mensaje `Hello World !` en pantalla . Esta vez, en vez imprimir el mensaje en pantalla, este será escrito en un archivo, el cual será escogido por el usuario.

El *plugin* contará con tres ventanas que se desplegaran, en forma consecutiva, a medida que se haga clic en los botones citados anteriormente.

Estas son las tres ventanas que desplegará el *plugin*:

Presentación

Presentación del *plugin*

Escoger un archivo

Escoger el archivo sobre el que se hará la esctritura de la frase "Hello World"

Mensaje final

Mensaje indicando que el proceso ha sido completado

El *plugin* completo lo puede encontrar en el siguiente archivo: `HelloWorld.jar` (HelloWorld2/HelloWorld.jar)

Para agregarle mayor funcionalidad al *plugin* se modificó la clase `HelloWorld` y se crearon nuevas clases. El proceso completo será explicado a continuación:

3.1. Modificar la clase `HelloWorld`

Debido a que deseamos manejar el despliegue visual del *plugin* al estilo de un *"wizard"*, modificaremos la clase `HelloWorld` para que su única responsabilidad sea crear la clase `WizardManager`, a la cual delegaremos la responsabilidad del manejo de ventanas del *plugin*.

El codigo final de esta clase quedó de la siguiente manera:

```
import org.gjt.sp.jedit.*;

 public class HelloWorld
 {
         public HelloWorld(View view)  {
       WizardManager wizardManager = new WizardManager(view);
     }
 }
```

El código de esta clase lo puede encontrar en:

3.2. Creación de la clase `WizardManager`

Esta clase será la encargada de desplegar la visualización correspondiente al paso de la tarea en que se encuentre el *plugin*.

La información sobre el paso de la tarea donde se encuentra el *plugin* será almacenada en una variable de la clase llamada `panelNumber`. Esta variable podrá ser modificada mediante los métodos next() y back(), dependiendo de si se quiere avanzar o retroceder un paso.

El código completo de esta clase lo puede encontrar en:

La variable `panelNumber` será inicializada en 0, cuando se cree por primera vez el objeto `WizardManager`. Los metodos next() y back() se encargaran de incrementar o decrementar esta variable y luego mostrar el panel correspondiente. La implementación de estos métodos es la siguiente:

```
public void back() {
   panelNumber--;
   showPanel();
}

public void next() {
   panelNumber++;
   showPanel();
}
```

El método showPanel() es el encargado de mostrar el panel correspondiente al paso de la tarea en que se encuentra el *plugin*. Este esta implementado así:

```
public void showPanel() {
   JPanel content=null;

   if(panelNumber==1)
 content = (JPanel) new HelloWorldPanel1();
   if(panelNumber==2)
 content = (JPanel) new HelloWorldPanel2();
   if(panelNumber==3)
 content = (JPanel) new HelloWorldPanel3();

   setContentPane(content);
   this.pack();
   this.repaint();
   this.show();
}
```

Buscando obtener una mayor flexibilidad, facilidad de uso y claridad al hacer los llamados a los métodos de la clase `WizardManager` desde otras clases, se le aplicó a esta clase un patrón de construcción de software conocido como *Singleton*, que permite hacer llamados a los métodos de esta clase desde otras clases, como los paneles, sin que estas otras clases deban tener como atributo el objeto `WizardManager`. Para ello basta usar una línea como la siguiente

```
WizardManager.getInstance().metodo();
```

3.3. Creación de la clase `HelloWorldPanel1`

Esta clase contiene el panel correspondiente al primer paso del *plugin*. Aquí se especifica tanto el componente gráfico del panel como su manejo de eventos.

El código completo de esta clase lo puede encontrar en:

Para este panel, dado que es el primero, el botón "Anterior" será desactivado.

Dentro del manejo de eventos de este panel se harán llamados a métodos de la clase `WizardManager`. Por ejemplo, cuando se presiona el botón "Siguiente >" se llamará el método next() de la siguiente manera:

```
WizardManager.getInstance().next();
```

3.4. Creación de la clase `HelloWorldPanel2`

Esta clase manejará el panel correspondiente al segundo paso del *plugin*, que se encarga de escoger el archivo en el que se hará la escritura.

El código completo de esta clase lo puede encontrar en:

Para la selección del archivo utilizaremos la clase `JFIleChooser` de Java, la cual nos permitirá desplegar un cuadro de diálogo de selección de archivos y luego de que el usuario seleccione el archivo, podremos obtener las características del archivo seleccionado.

Al presionar el botón "Examinar..." , se realizará el siguiente manejo del evento, que incluye el despliegue del cuadro de diálogo de selección de archivos y la obtención del archivo seleccionado:

```
void jButton4_actionPerformed(ActionEvent e) {
    JFileChooser jFileChooser1 = new JFileChooser();
    jFileChooser1.setDialogTitle("Ubicación del archivo resultante");

    int returnVal = jFileChooser1.showSaveDialog(WizardManager.getInstance());

    if (returnVal == JFileChooser.APPROVE_OPTION) {
 resultFile = jFileChooser1.getSelectedFile();
 String path = new String (resultFile.getPath().toString());
this.jTextField1.setText(path);
    } else {
jFileChooser1.setVisible(false);
    }
}
```

Luego, para imprimir en el archivo seleccionado se utilizará el método que posee el siguiente código:

```
public void printContentInFile(){
    String path = resultFile.getPath();
    FileWriter fw = null;
    PrintWriter pw = null;

    try {
 File out = new File(path);
 fw = new FileWriter(out);
 pw = new PrintWriter(fw);
    }
    catch(Exception e){
 JOptionPane.showMessageDialog(WizardManager.getInstance(),
          e.getMessage(),
          "Error", JOptionPane.ERROR_MESSAGE);
    }

    pw.print("Hello World !");
```

```
    try {
 pw.close();
 fw.close();
    }
    catch(Exception e){
 JOptionPane.showMessageDialog(WizardManager.getInstance(),
          e.getMessage(),
          "Error", JOptionPane.ERROR_MESSAGE);
    }
}
```

3.5. Creación de la clase `HelloWorldPanel3`

Por último, la clase `HelloWorldPanel3` manejará el panel correspondiente al paso final del *plugin*. Este panel solo estará encargado de desplegar el mensaje correspondiente al final del proceso.

El código completo de esta clase lo puede encontrar en:

Apéndice A.

A.1. Listado de archivos del *plugin Hello World*

A.1.1. Archivo `HelloWorldPlugin.java`

```
import org.gjt.sp.jedit.*;

import java.util.Vector;

public class HelloWorldPlugin extends EditPlugin
{
 public void createMenuItems(Vector menuItems)
 {
  menuItems.addElement(GUIUtilities.loadMenuItem("HelloWorld"));
 }
}
```

A.1.2. Archivo `HelloWorld.java`

```
iimport javax.swing.*;
import org.gjt.sp.jedit.*;

public class HelloWorld extends JDialog
{
        // private members
        private View view;

        public HelloWorld(View view)
        {
                super(view,jEdit.getProperty("HelloWorld.title"),false);

                this.view = view;

                JPanel content = new JPanel();
                setContentPane(content);

                JLabel caption = new JLabel(jEdit.getProperty("HelloWorld.caption"));
                content.add(caption);

                pack();
                GUIUtilities.loadGeometry(this, "HelloWorld");
```

```
                setDefaultCloseOperation(DISPOSE_ON_CLOSE);
                show();
        }


        public void dispose()
        {
                view = null;
                GUIUtilities.saveGeometry(this, "HelloWorld");
                super.dispose();
        }
}
```

A.1.3. Archivo `actions.xml`

```
<?xml version="1.0"?>

<!DOCTYPE ACTIONS SYSTEM "actions.dtd">

<ACTIONS>
<ACTION NAME="HelloWorld">
<CODE>
new HelloWorld(view);
</CODE>
</ACTION>
</ACTIONS>
```

A.1.4. Archivo `HelloWorld.props`

```
# Plugin properties
plugin.HelloWorldPlugin.name=Hello World !
plugin.HelloWorldPlugin.author=Jairo Martinez
plugin.HelloWorldPlugin.version=1.0
plugin.HelloWorldPlugin.docs=HelloWorld.html

# Menu item label
HelloWorld.label=Hello World !

# HelloWorld window
HelloWorld.title=Hello World plugin
HelloWorld.caption=Hello World !
```

A.2. Listado de archivos del *plugin Hello World* modificado para agregarle mayor funcionalidad

A.2.1. Archivo `HelloWorld.java`

```
import org.gjt.sp.jedit.*;

public class HelloWorld
{
  public HelloWorld(View view)  {
    WizardManager wizardManager = new WizardManager(view);
  }
}
```

A.2.2. Archivo `WizardManager.java`

```
import javax.swing.*;
import org.gjt.sp.jedit.*;

public class WizardManager extends JDialog  {

  private View view;
  private int panelNumber = 0;
  private static WizardManager instance = null;

  public WizardManager(View view) {
    super(view,jEdit.getProperty("HelloWorld.title"),false);
    this.view = view;
    this.instance = this;
    GUIUtilities.loadGeometry(this, "HelloWorld");
    setDefaultCloseOperation(DISPOSE_ON_CLOSE);
    next();
  }

  public static  WizardManager getInstance () {
    return instance;
  }

  public void dispose() {
```

```
    view = null;
    GUIUtilities.saveGeometry(this, "HelloWorld");
    super.dispose();
  }

  public void back()  {
    panelNumber--;
    showPanel();
  }

  public void next() {
    panelNumber++;
    showPanel();
  }

  public void showPanel()  {
    JPanel content=null;

    if(panelNumber==1)
      content = (JPanel) new HelloWorldPanel1();
    if(panelNumber==2)
      content = (JPanel) new HelloWorldPanel2();
    if(panelNumber==3)
      content = (JPanel) new HelloWorldPanel3();

    setContentPane(content);
    this.pack();
    this.repaint();
    this.show();
  }

}
```

A.2.3. Archivo `HelloWorldPanel1.java`

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class HelloWorldPanel1 extends JPanel {
  BorderLayout borderLayout1 = new BorderLayout();
  JPanel jPanel1 = new JPanel();
  JPanel jPanel2 = new JPanel();
  JButton jButton1 = new JButton();
  JButton jButton2 = new JButton();
  JButton jButton3 = new JButton();
  GridLayout gridLayout1 = new GridLayout();
  JLabel jLabel1 = new JLabel();
```

```
JLabel jLabel2 = new JLabel();
JLabel jLabel3 = new JLabel();
JLabel jLabel4 = new JLabel();

public HelloWorldPanel1() {
  try {
    jbInit();
  }
  catch(Exception ex) {
    ex.printStackTrace();
  }
}

void jbInit() throws Exception {
  this.setLayout(borderLayout1);
  jButton1.setEnabled(false);
  jButton1.setText("< Anterior");
  jButton1.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(ActionEvent e) {
      jButton1_actionPerformed(e);
    }
  });
  jButton2.setText("Siguiente >");
  jButton2.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(ActionEvent e) {
      jButton2_actionPerformed(e);
    }
  });
  jButton3.setText("Cancelar");
  jButton3.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(ActionEvent e) {
      jButton3_actionPerformed(e);
    }
  });
  jPanel1.setLayout(gridLayout1);
  jLabel1.setFont(new java.awt.Font("Dialog", 1, 16));
  jLabel1.setHorizontalAlignment(SwingConstants.CENTER);
  jLabel1.setText("Bienvenido al plugin Hello World !");
  jLabel2.setHorizontalAlignment(SwingConstants.CENTER);
  jLabel2.setText("Este plugin le permitira escribir en un archivo");
  gridLayout1.setRows(4);
  gridLayout1.setColumns(1);
  jLabel3.setHorizontalAlignment(SwingConstants.CENTER);
  jLabel3.setText("escogido la frase \"Hello World !\"");
  jLabel4.setHorizontalAlignment(SwingConstants.CENTER);
  jLabel4.setText("Presione el boton \"Siguiente >\" para continuar");
  this.add(jPanel1,  BorderLayout.CENTER);
  jPanel1.add(jLabel1, null);
  jPanel1.add(jLabel2, null);
  jPanel1.add(jLabel3, null);
  jPanel1.add(jLabel4, null);
  this.add(jPanel2, BorderLayout.SOUTH);
  jPanel2.add(jButton1, null);
```

```
    jPanel2.add(jButton2, null);
    jPanel2.add(jButton3, null);

  }

  void jButton1_actionPerformed(ActionEvent e) {
    WizardManager.getInstance().back();
  }

  void jButton2_actionPerformed(ActionEvent e) {
    WizardManager.getInstance().next();
  }

  void jButton3_actionPerformed(ActionEvent e) {
     WizardManager.getInstance().dispose();
  }
}s.CENTER);
    jLabel3.setText("escogido la frase \"Hello World !\"");
    jLabel4.setHorizontalAlignment(SwingConstants.CENTER);
    jLabel4.setText("Presione el boton \"Siguiente >\" para continuar");
    this.add(jPanel1,  BorderLayout.CENTER);
    jPanel1.add(jLabel1, null);
    jPanel1.add(jLabel2, null);
    jPanel1.add(jLabel3, null);
    jPanel1.add(jLabel4, null);
    this.add(jPanel2, BorderLayout.SOUTH);
    jPanel2.add(jButton1, null);
    jPanel2.add(jButton2, null);
    jPanel2.add(jButton3, null);

  }

  void jButton1_actionPerformed(ActionEvent e) {
    WizardManager.getInstance().back();
  }

  void jButton2_actionPerformed(ActionEvent e) {
    WizardManager.getInstance().next();
  }

  void jButton3_actionPerformed(ActionEvent e) {
     WizardManager.getInstance().dispose();
  }
}
```

A.2.4. Archivo `HelloWorldPane2.java`

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.io.*;

public class HelloWorldPanel2 extends JPanel {
  File resultFile;
  BorderLayout borderLayout1 = new BorderLayout();
  JPanel jPanel1 = new JPanel();
  JPanel jPanel2 = new JPanel();
  JButton jButton1 = new JButton();
  JButton jButton2 = new JButton();
  JButton jButton3 = new JButton();
  GridLayout gridLayout1 = new GridLayout();
  JLabel jLabel1 = new JLabel();
  JLabel jLabel2 = new JLabel();
  JPanel jPanel3 = new JPanel();
  JButton jButton4 = new JButton();
  JLabel jLabel3 = new JLabel();
  JTextField jTextField1 = new JTextField();
  JLabel jLabel4 = new JLabel();

  public HelloWorldPanel2() {
    try {
      jbInit();
    }
    catch(Exception ex) {
      ex.printStackTrace();
    }
  }

  void jbInit() throws Exception {
    this.setLayout(borderLayout1);
    jButton1.setText("< Anterior");
    jButton1.addActionListener(new java.awt.event.ActionListener() {
      public void actionPerformed(ActionEvent e) {
        jButton1_actionPerformed(e);
      }
    });
    jButton2.setText("Siguiente >");
    jButton2.addActionListener(new java.awt.event.ActionListener() {
      public void actionPerformed(ActionEvent e) {
        jButton2_actionPerformed(e);
      }
    });
    jButton3.setText("Cancelar");
    jButton3.addActionListener(new java.awt.event.ActionListener() {
      public void actionPerformed(ActionEvent e) {
        jButton3_actionPerformed(e);
```

```
    }
  });
  jPanel1.setLayout(gridLayout1);
  jLabel1.setFont(new java.awt.Font("Dialog", 1, 16));
  jLabel1.setToolTipText("");
  jLabel1.setHorizontalAlignment(SwingConstants.CENTER);
  jLabel1.setText("Escoja el archivo");
  jLabel2.setHorizontalAlignment(SwingConstants.CENTER);
  jLabel2.setText("Presione el boton \"Examinar...\" para escoger");
  gridLayout1.setRows(3);
  gridLayout1.setColumns(1);
  jButton4.setText("Examinar...");
  jButton4.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(ActionEvent e) {
      jButton4_actionPerformed(e);
    }
  });
  jLabel3.setToolTipText("");
  jLabel3.setHorizontalAlignment(SwingConstants.CENTER);
  jLabel3.setText("el nombre y ubicación del archivo");
  jTextField1.setColumns(15);
  jLabel4.setText("Ubicación");
  this.add(jPanel1, BorderLayout.NORTH);
  jPanel1.add(jLabel1, null);
  jPanel1.add(jLabel2, null);
  jPanel1.add(jLabel3, null);
  this.add(jPanel2,  BorderLayout.SOUTH);
  jPanel2.add(jButton1, null);
  jPanel2.add(jButton2, null);
  jPanel2.add(jButton3, null);
  this.add(jPanel3,  BorderLayout.CENTER);
  jPanel3.add(jLabel4, null);
  jPanel3.add(jTextField1, null);
  jPanel3.add(jButton4, null);

}

void jButton1_actionPerformed(ActionEvent e) {
  WizardManager.getInstance().back();
}

void jButton2_actionPerformed(ActionEvent e) {
  printContentInFile();
  WizardManager.getInstance().next();
}

void jButton3_actionPerformed(ActionEvent e) {
   WizardManager.getInstance().dispose();
}

void jButton4_actionPerformed(ActionEvent e) {
  JFileChooser jFileChooser1 = new JFileChooser();
  jFileChooser1.setDialogTitle("Ubicación del archivo resultante");
```

```
    int returnVal = jFileChooser1.showSaveDialog(WizardManager.getInstance());
    if (returnVal == JFileChooser.APPROVE_OPTION) {
      resultFile = jFileChooser1.getSelectedFile();
      String path = new String (resultFile.getPath().toString());
      this.jTextField1.setText(path);
    } else {
       jFileChooser1.setVisible(false);
    }
  }

  public void printContentInFile(){
    String path = resultFile.getPath();
    FileWriter fw = null;
    PrintWriter pw = null;

    try {
      File out = new File(path);
      fw = new FileWriter(out);
      pw = new PrintWriter(fw);
    }
    catch(Exception e){
      JOptionPane.showMessageDialog(WizardManager.getInstance(), e.getMessage(), "Er-
ror", JOptionPane.ERROR_MESSAGE);
    }

    pw.print("Hello World !");

    try {
          pw.close();
          fw.close();
    }
    catch(Exception e){
      JOptionPane.showMessageDialog(WizardManager.getInstance(), e.getMessage(), "Er-
ror", JOptionPane.ERROR_MESSAGE);
    }
  }
}
```

A.2.5. Archivo `HelloWorldPane3.java`

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class HelloWorldPanel3 extends JPanel {
  BorderLayout borderLayout1 = new BorderLayout();
  JPanel jPanel1 = new JPanel();
  JPanel jPanel2 = new JPanel();
```

```
JButton jButton1 = new JButton();
JButton jButton2 = new JButton();
JButton jButton3 = new JButton();
GridLayout gridLayout1 = new GridLayout();
JLabel jLabel1 = new JLabel();
JLabel jLabel2 = new JLabel();
JLabel jLabel3 = new JLabel();
JLabel jLabel4 = new JLabel();

public HelloWorldPanel3() {
  try {
    jbInit();
  }
  catch(Exception ex) {
    ex.printStackTrace();
  }
}

void jbInit() throws Exception {
  this.setLayout(borderLayout1);
  jButton1.setText("< Anterior");
  jButton1.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(ActionEvent e) {
      jButton1_actionPerformed(e);
    }
  });
  jButton2.setText("Terminar");
  jButton2.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(ActionEvent e) {
      jButton2_actionPerformed(e);
    }
  });
  jButton3.setText("Cancelar");
  jButton3.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(ActionEvent e) {
      jButton3_actionPerformed(e);
    }
  });
  jPanel1.setLayout(gridLayout1);
  jLabel1.setFont(new java.awt.Font("Dialog", 1, 16));
  jLabel1.setHorizontalAlignment(SwingConstants.CENTER);
  jLabel1.setText("Gracias por utilizar el plugin Hello World !");
  jLabel2.setHorizontalAlignment(SwingConstants.CENTER);
  jLabel2.setText("La frase \"Hello World !\" ha quedado escrita");
  gridLayout1.setRows(4);
  gridLayout1.setColumns(1);
  jLabel3.setHorizontalAlignment(SwingConstants.CENTER);
  jLabel3.setText("en el archivo que usted ha escogido. ");
  jLabel4.setToolTipText("");
  jLabel4.setHorizontalAlignment(SwingConstants.CENTER);
  this.add(jPanel1,  BorderLayout.CENTER);
  jPanel1.add(jLabel1, null);
  jPanel1.add(jLabel2, null);
```

```
    jPanel1.add(jLabel3, null);
    jPanel1.add(jLabel4, null);
    this.add(jPanel2, BorderLayout.SOUTH);
    jPanel2.add(jButton1, null);
    jPanel2.add(jButton2, null);
    jPanel2.add(jButton3, null);

  }

  void jButton1_actionPerformed(ActionEvent e) {
    WizardManager.getInstance().back();
  }

  void jButton2_actionPerformed(ActionEvent e) {
    WizardManager.getInstance().dispose();
  }

  void jButton3_actionPerformed(ActionEvent e) {
     WizardManager.getInstance().dispose();
  }
}
```