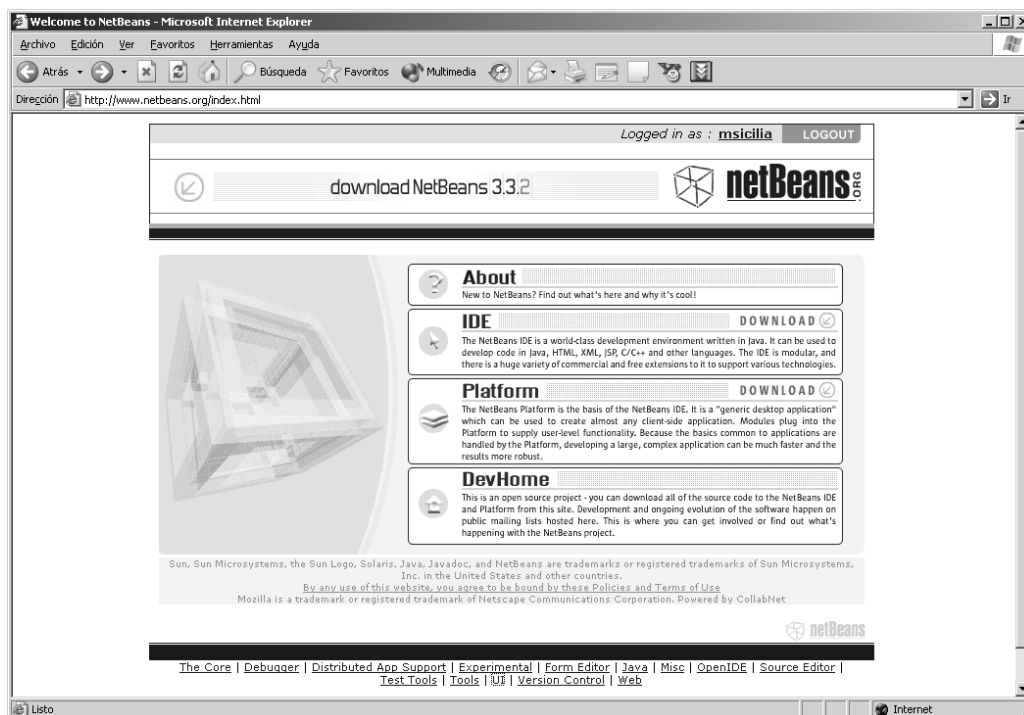




# El Entorno de Programación de Fuente Abierto NetBeans versión 3.3.2





## INDICE

1. Introducción a NetBeans.....	3
2. Montar el Sistema de Ficheros.....	4
3. Una Primera Aplicación con NetBeans.....	5
4. Uso del Gestor de Proyectos.....	15
5. Depurar una Aplicación .....	17
6. Direcciones Web.....	19
7. Soluciones a las Actividades y Ejercicios .....	19
7.1. Actividad Propuesta 1.....	19



## 1. Introducción a NetBeans

El Entorno de Desarrollo Integrado (IDE) **NetBeans** es un entorno de programación para varios lenguajes, incluyendo a Java y C++. Este desarrollo es de fuente abierto, es decir, se proporciona el código fuente del entorno para que se pueda modificar de acuerdo a ciertos parámetros de licencia. Para más detalles, se puede consultar:

<http://www.opensource.org/docs/osd-spanish.html>

Nota: NetBeans es también una **plataforma de ejecución** de aplicaciones, es decir, facilita la escritura de aplicaciones Java, proporcionando una serie de *servicios* comunes, que a su vez están disponibles a través del IDE. En este documento, nos centramos en el IDE como aplicación, sin entrar en detalles de esa plataforma de ejecución.

Desde la página de NetBeans se puede descargar el entorno, y acceder a su documentación, foros y otros recursos. Más concretamente, tenemos una página de recursos relacionados con la programación de interfaz gráfica en:

<http://ui.netbeans.org/>

El entorno requiere una instalación separada del JDK de Java (la intenta detectar automáticamente al instalarse).

Al utilizarse por primera vez, nos solicita que especifiquemos un directorio en el cual se almacenarán nuestras configuraciones y proyectos:

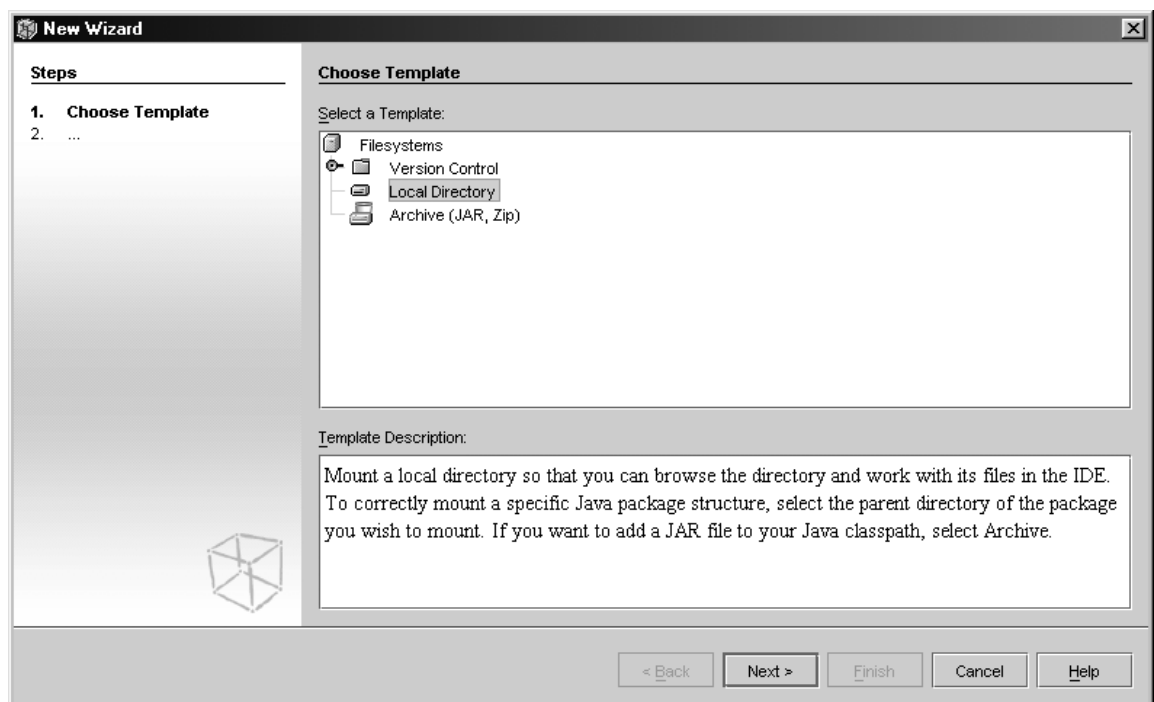




En este documento introducimos los conceptos fundamentales del entorno. En la ayuda integrada en el IDE contamos con una sección "Getting Started" donde se describe con mayor detalle.

## 2. Montar el Sistema de Ficheros

Para poder empezar a trabajar con NetBeans necesitamos tener mapeado nuestro directorio de trabajo. En el explorador de los Sistemas de Ficheros encontraremos dichos directorios. Si deseamos añadir un nuevo directorio, se puede utilizar la opción *File|Mount Filesystem*. Eso hará que aparezca el *Asistente* correspondiente.

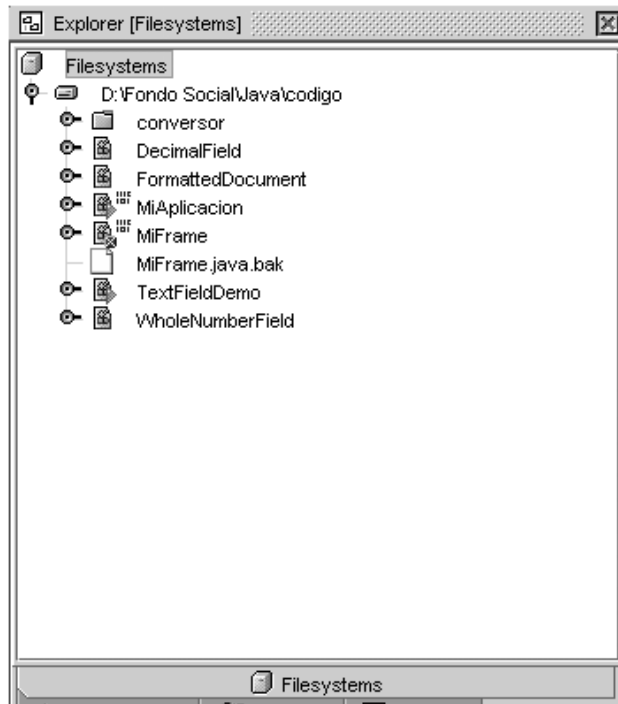


Deberemos seleccionar la opción *Local Directory* si queremos mapear un directorio local y a continuación pulsar en *Next*. También nos permite las opciones de conseguir un directorio controlado por CVS, o un archivo que este comprimido con JAR o Zip.

A continuación nos aparecerá un explorador de directorios de nuestros discos locales, sobre el cual deberemos navegar hasta encontrar aquel directorio en el que vamos a trabajar. Una vez seleccionado pulsar *Finish*. A

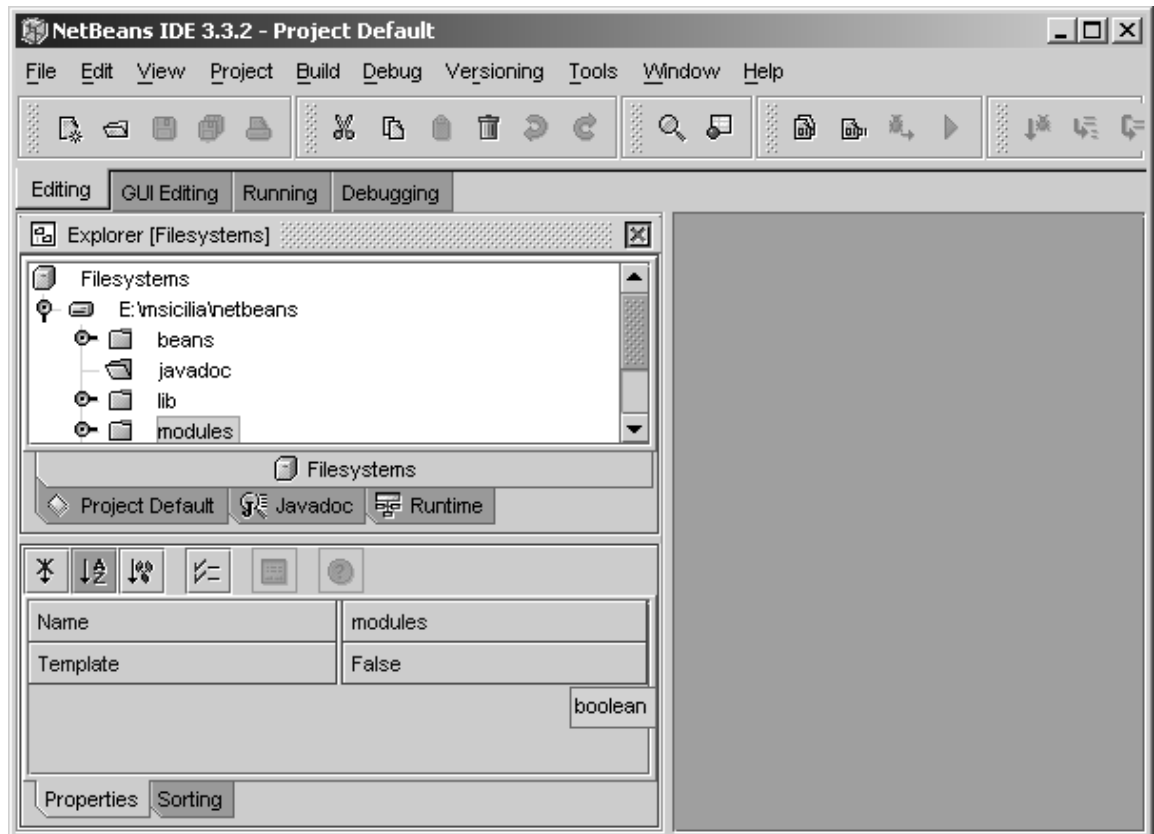


continuación podremos visualizar dicho directorio y su contenido desde el explorador del Sistema de Ficheros.



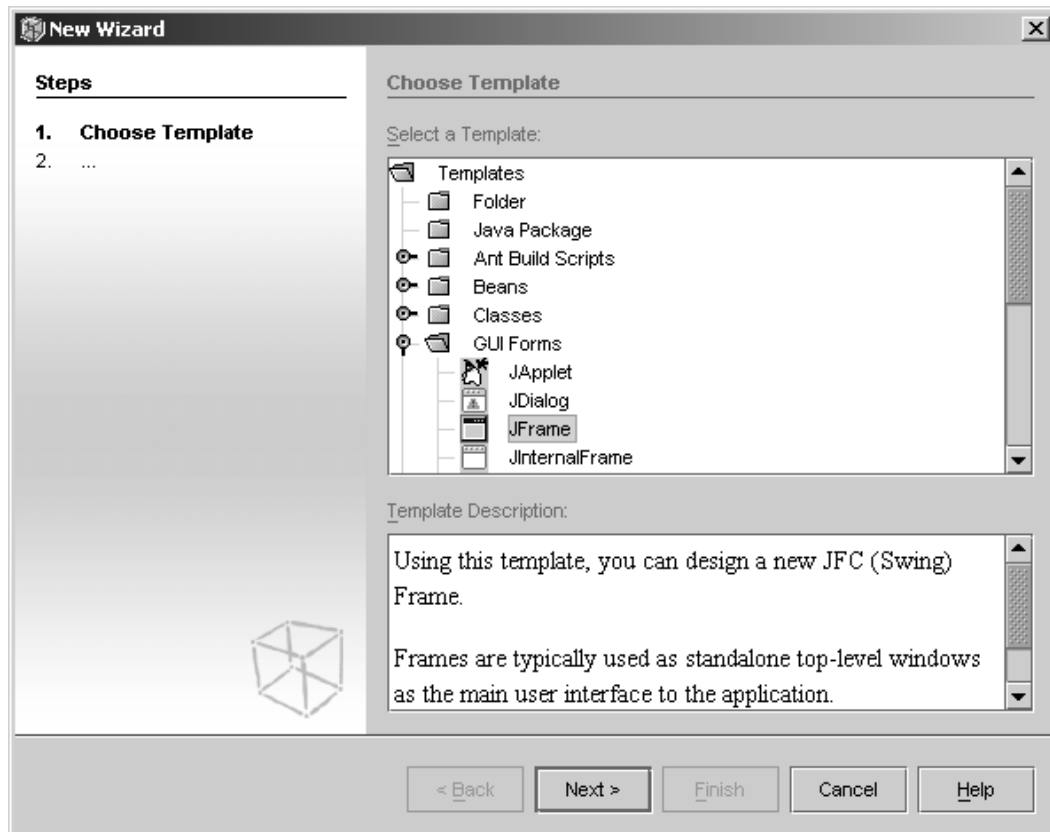
### 3. Una Primera Aplicación con NetBeans

Quando abrimos el entorno de NetBeans (quizá se abra un diálogo de bienvenida que podemos cerrar), el entorno tendrá una apariencia como la siguiente:

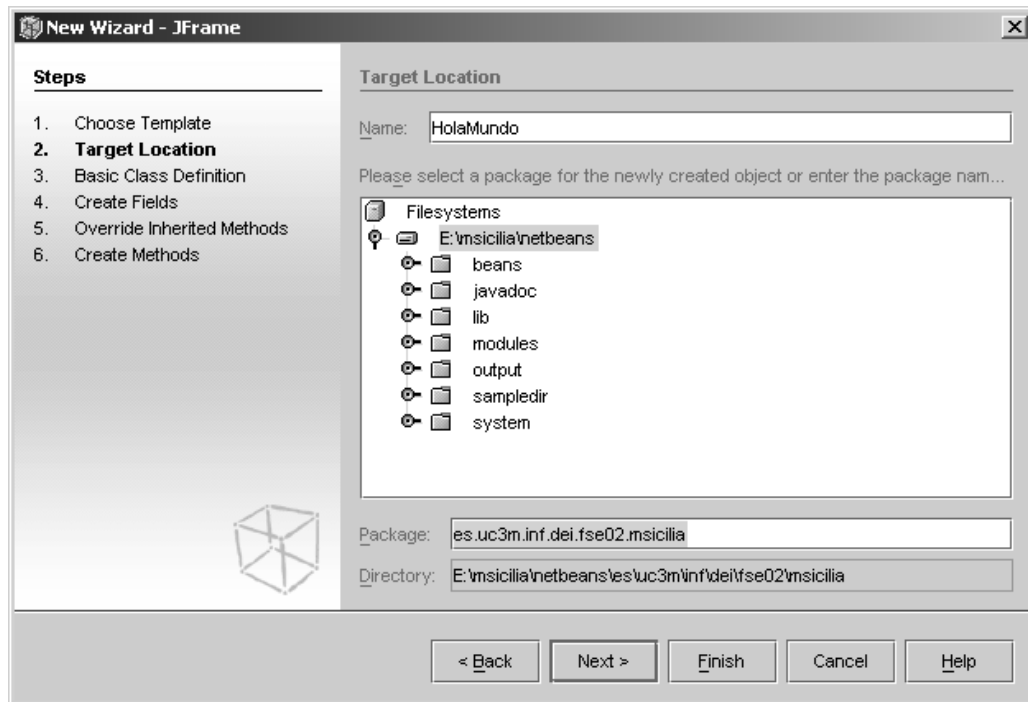


Vemos básicamente el explorador de los Sistemas de Ficheros, donde tendremos los contenidos de nuestro directorio local.

Para comenzar a crear una aplicación, la manera más simple es utilizar la opción *File|New* y seleccionar una de las plantillas (*Templates*) que nos proporciona NetBeans. Concretamente, seleccionaremos **Jframe**, que se encuentra dentro de la carpeta *GUI Forms*, para crear una aplicación Swing vacía. Eso hará que aparezca el *Asistente* correspondiente.

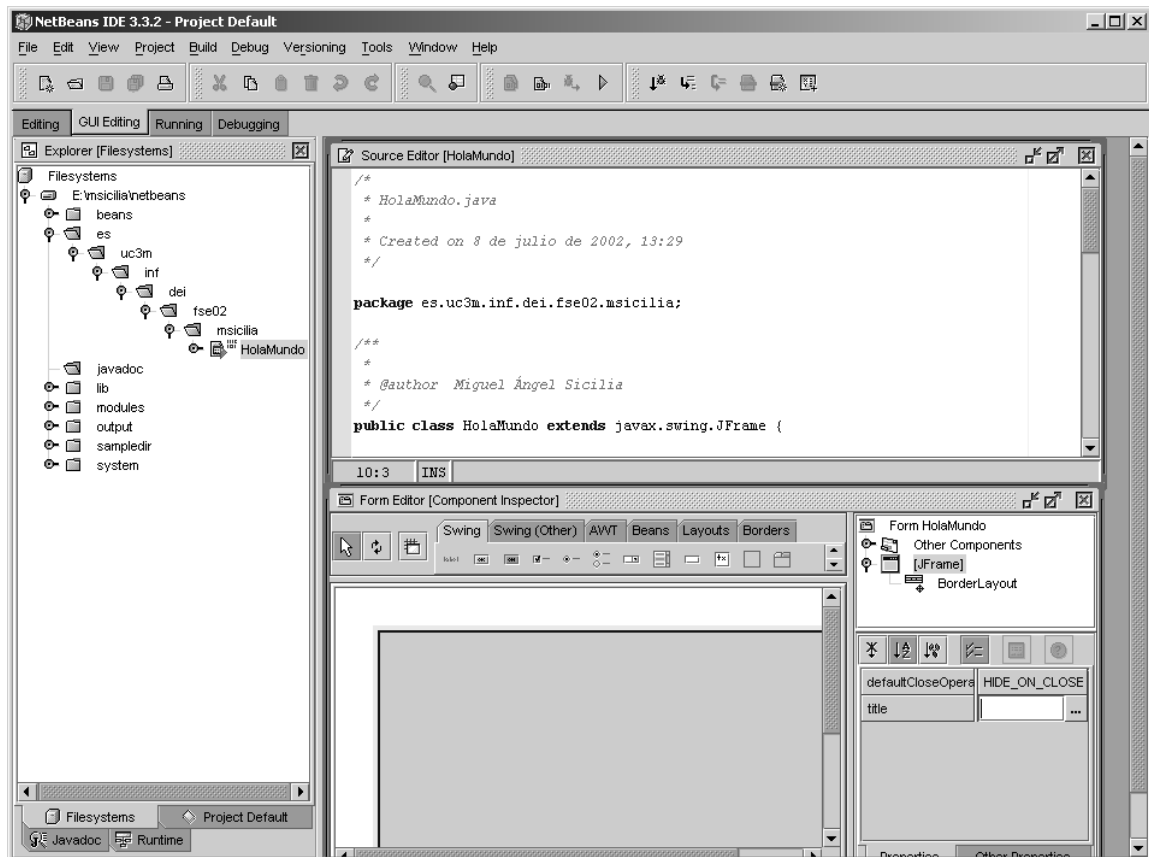


A continuación tendremos que indicar el paquete (y su ubicación física en un sistema de ficheros) en el que queremos que se incluya el código que se va a generar automáticamente.



A continuación nos permitirá indicar los detalles de la clase (su visibilidad, la clase de la que hereda, etc.) así como generar código para atributos o métodos nuevos o redefinidos de su superclase. De momento, sencillamente pulsaremos "Next" ya que todo eso se puede hacer con posterioridad desde el entorno. Nótese que en la última de las pantallas del asistente, vemos que NetBeans va a incluir tres métodos en nuestra clase derivada de JFrame:  *initComponents* ,  *exitForm*  y  *main* .

Se abrirán dos ventanas en el entorno, como se muestra a continuación:



Nótese dónde ha ubicado la nueva clase dentro del sistema de ficheros que seleccionamos, y también que el entorno se encuentra en modo "GUI Editing" (véase la solapa en la parte superior derecha, bajo la barra de botones). A estos modos que se activan con esas solapas, se les denomina "Workspaces" en NetBeans.

Por defecto, se abren dos ventanas: La de edición del código fuente (*source*) y la de edición gráfica (*form*). Vamos a examinarlas en más detalle.

El código fuente generado incluye los métodos que vimos antes, junto con un constructor. Es importante tener en cuenta que en este editor aparecen **fragmentos de código sombreados que no deben modificarse**, ya que están bajo el control del entorno NetBeans. Por ejemplo, el método *initComponents*, que se invoca desde el constructor.



Ya podemos utilizar la opción *Build|Execute* para ejecutar la clase que acabamos de generar. Veremos que nos aparece una ventana con el *Frame*, y la única cosa que podemos hacer con ella es cerrarla, pulsando el botón de cierre de la ventana, que hará que pasemos por el código de *exitForm*.

Nótese que si seleccionamos la solapa de *"Running"*, en la parte superior derecha, aparecerá nuestra aplicación en la *"Execution View"*, y también la denominada *"Output Window"* que equivale a la consola de ejecución donde aparecen los *System.out* que podamos haber incluido en el código.

**Actividad:** Incluir un mensaje con *System.out* en el constructor del *Frame*, volver a ejecutar, y observar la *Output Window*.

Las opciones de los menús *"View"* y *"Windows"* sirven para personalizar la apariencia del entorno. Podemos probar sus efectos.

La vista de edición gráfica (*Form*) nos permite incluir elementos de GUI seleccionándolos de una paleta. En primer lugar, tenemos que observar el **inspector de Componentes**, que nos muestra una vista jerárquica de los componentes en el formulario, así como las propiedades modificables de cada uno de ellos. Por defecto, nos ha creado un *BorderLayout* como gestor de posición, pero podemos cambiarlo, por ejemplo, por un *FlowLayout*, utilizando el menú contextual del gestor de contenido. Probemos ahora a incluir un componente de la paleta, por ejemplo, un *JToggleButton*. Basta con arrastrarlo y soltarlo sobre el formulario, y se añadirá al inspector de componentes. Desde ese editor, podremos cambiar sus propiedades, como por ejemplo el texto que muestra. También podremos añadir eventos.

**Actividad:** Añadir un evento al botón de manera que cambie su texto de *"Pulsado"* a *"No Pulsado"*. Para ello, se puede utilizar la solapa *"Events"* y hacer clic sobre el evento deseado, lo cual generará un esqueleto por defecto, que podemos editar desde la ventana de edición del código fuente. Esto puede hacerse con los métodos *isSelected* y *setText* de *JToggleButton*. Para ayudarnos en la escritura, podemos utilizar la característica de NetBeans que intenta completar el código que vamos haciendo. Si no se activa sola, se puede acceder a ella con *ctrl+Espacio* sobre una variable en el editor de código.



Vamos ahora a crear un cuadro de diálogo que se muestre cuando intentemos cerrar la aplicación, pidiendo confirmación de si realmente queremos salir. Para ello, utilizaremos el correspondiente *template* **OK/Cancel JDialog**, que se encuentra en *GUI Forms|Sample Forms* y le llamaremos *DialogoSalir*. Una parte del código generado es el siguiente:

```
public class DialogoSalir extends javax.swing.JDialog {
/** A return status code - returned if Cancel button has been pressed */
    public static final int RET_CANCEL = 0;
/** A return status code - returned if OK button has been pressed */
    public static final int RET_OK = 1;
/** Creates new form DialogoSalir */
    public DialogoSalir(java.awt.Frame parent, boolean modal) {
        super(parent, modal);
        initComponents();
    }
/** This method is called from within the constructor to
 * initialize the form.
 * WARNING: Do NOT modify this code. The content of this method is
 * always regenerated by the Form Editor.
 */
    private void initComponents() {
```



```
        buttonPanel = new javax.swing.JPanel();
        okButton = new javax.swing.JButton();
        cancelButton = new javax.swing.JButton();
        addWindowListener(new java.awt.event.WindowAdapter() {
            public void windowClosing(java.awt.event.WindowEvent evt) {
                closeDialog(evt);
            }
        });

        buttonPanel.setLayout(new
java.awt.FlowLayout(java.awt.FlowLayout.RIGHT));

        okButton.setText("OK");
        okButton.addActionListener(new
java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent
evt) {
                okButtonActionPerformed(evt);
            }
        });

        buttonPanel.add(okButton);

        cancelButton.setText("Cancel");
        cancelButton.addActionListener(new
java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent
evt) {
                cancelButtonActionPerformed(evt);
            }
        });

        buttonPanel.add(cancelButton);

        getContentPane().add(buttonPanel,
java.awt.BorderLayout.SOUTH);

        pack();
    }

    private void okButtonActionPerformed(java.awt.event.ActionEvent evt) {
        doClose (RET_OK);
    }

    private void cancelButtonActionPerformed(java.awt.event.ActionEvent evt)
    {
        doClose (RET_CANCEL);
    }
}
```



```
/** Closes the dialog */
private void closeDialog(java.awt.event.WindowEvent evt) {
    doClose (RET_CANCEL);
}

private void doClose(int retStatus) {
    returnStatus = retStatus;
    setVisible(false);
    dispose();
}

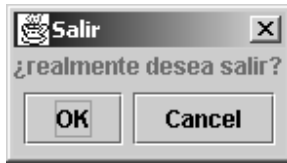
// Variables declaration - do not modify
private javax.swing.JPanel buttonPanel;
private javax.swing.JButton okButton;
private javax.swing.JButton cancelButton;
// End of variables declaration

private int returnStatus = RET_CANCEL;
}
```

En el código podemos ver cómo se ha proporcionado un atributo privado *returnStatus* que guarda lo que el usuario pulsó en el cuadro de diálogo (codificado como un entero). Podemos ver también como se han codificado dos eventos para guardar en *returnStatus* lo que el usuario pulsó (mediante el método *doClose*).

Así, si queremos utilizar este código desde nuestro anterior *JPanel*, lo que tendremos que hacer es:

1. Añadir una etiqueta que nos pregunte "¿realmente desea salir?". Si nos fijamos en el código generado, tenemos que el diálogo está dentro de un Panel con un *BorderLayout*, que tiene en el "Sur" otro *JPanel*, que a su vez contiene los botones con un *FlowLayout*. Nosotros podemos poner en el "Norte" la etiqueta, de modo que quedará centrada respecto a los botones.
2. Crear una instancia del diálogo cuando se intente cerrar nuestra aplicación. Esto es tan sencillo como hacer que nuestro Panel sea el padre del diálogo, añadiéndolo al método *exitForm*.
3. Comprobar lo que el usuario pulsó y cerrar o no la aplicación dependiendo de ello.



El código que hay que añadir al frame HolaMundo es el siguiente:

```
private void exitForm(java.awt.event.WindowEvent evt) {  
    DialogoSalir d = new DialogoSalir(this, true);  
    d.show();  
    if (d.getReturnStatus()==DialogoSalir.RET_OK)  
        System.exit(0);  
}
```

Si añadimos el código anterior, veremos que aún así, el Frame sigue cerrándose (en realidad, se oculta, y el proceso sigue ejecutándose, como podemos ver en el "Execution View"). Esto es debido a que los *JFrame*, a diferencia de los Frame de AWT, por defecto ocultan la ventana. Para solventarlo, podemos añadir al constructor la siguiente sentencia (habiendo importado la interfaz *javax.swing.WindowConstants*):

```
setDefaultCloseOperation(WindowConstants.DO_NOTHING_ON_CLOSE);
```

**Actividad Propuesta 1:** Crear un menú contextual en el panel anterior que contenga dos opciones "Salir", que será igual que pulsar el botón de cerrar la ventana, y "Ayuda", que mostrará un simple cuadro de diálogo con el nombre del autor.

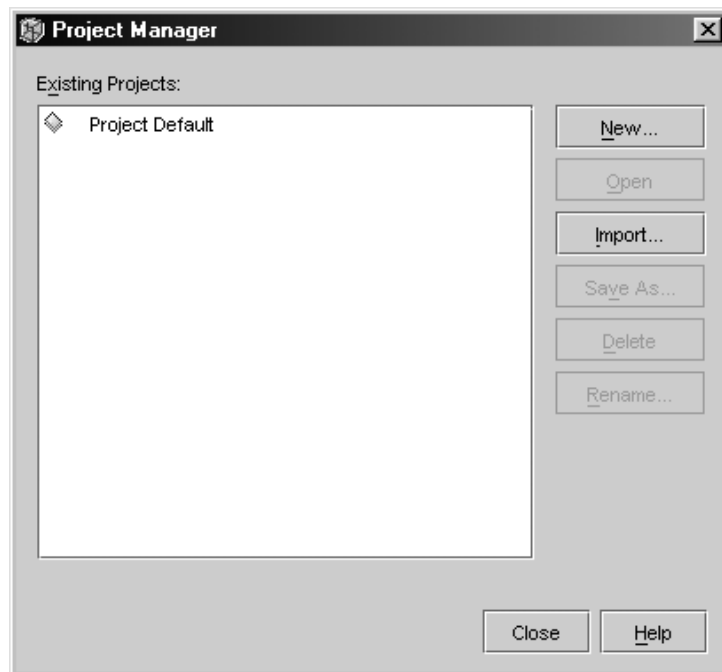
Para ello se debe recurrir a la documentación de la clase *JPopupMenu*. NetBeans nos permite construir el menú desde el editor de formularios. Para hacer que el menú aparezca cuando se pulse en el Frame,



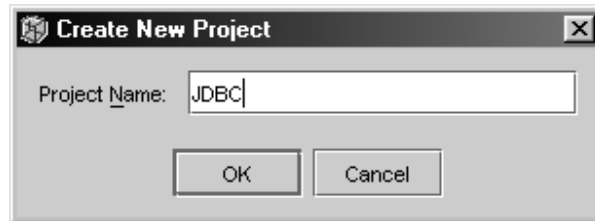
## 4. Uso del Gestor de Proyectos

En los ejemplos anteriores, no hemos utilizado las posibilidades del gestor de proyectos de NetBeans.

Para crear un proyecto que vaya a representar una aplicación que vamos a desarrollar seleccionaremos *Project|Project Manager*. Aparecerá una ventana que nos permitirá tanto crear un nuevo proyecto, como eliminar uno existente o importarlo.



En este caso seleccionaremos *New* para crear uno nuevo. En una nueva ventana le daremos un nombre.



Otras opciones que podemos encontrar

- *Set Project Main Class*. Si tenemos varios main en ese proyecto, podemos especificar cual será el principal.



- *Compile Project*. Todas las clases sin compilar y ficheros con fechas antiguas pertenecientes al proyecto, son compilados.
- *Buid Project*. Fuerza la compilación de todos los ficheros del proyecto.
- *Execute Project*. Una vez que hemos especificado cual es la clase que contiene el main, esta opción nos permitirá poner en ejecución el proyecto.
- *Debugging Project*. Nos permite ejecutar el proyecto en modo depuración (ver sección 5).



## 5. Depurar una Aplicación

Para poder depurar una aplicación es necesario poner un punto de ruptura o *BreakPoint*. Un breakpoint se añade poniendo el cursor sobre la línea de código donde queremos que la ejecución del programa se pare y seleccionar *Debug|Toggle Breakpoint*.

```
Source Editor [HolaMundo]
DialogSalir dialogo= new DialogSalir("OK",true);
dialogo.show();
if (dialogo.getReturnStatus()== DialogSalir.RET_OK)
    System.exit(0);
}

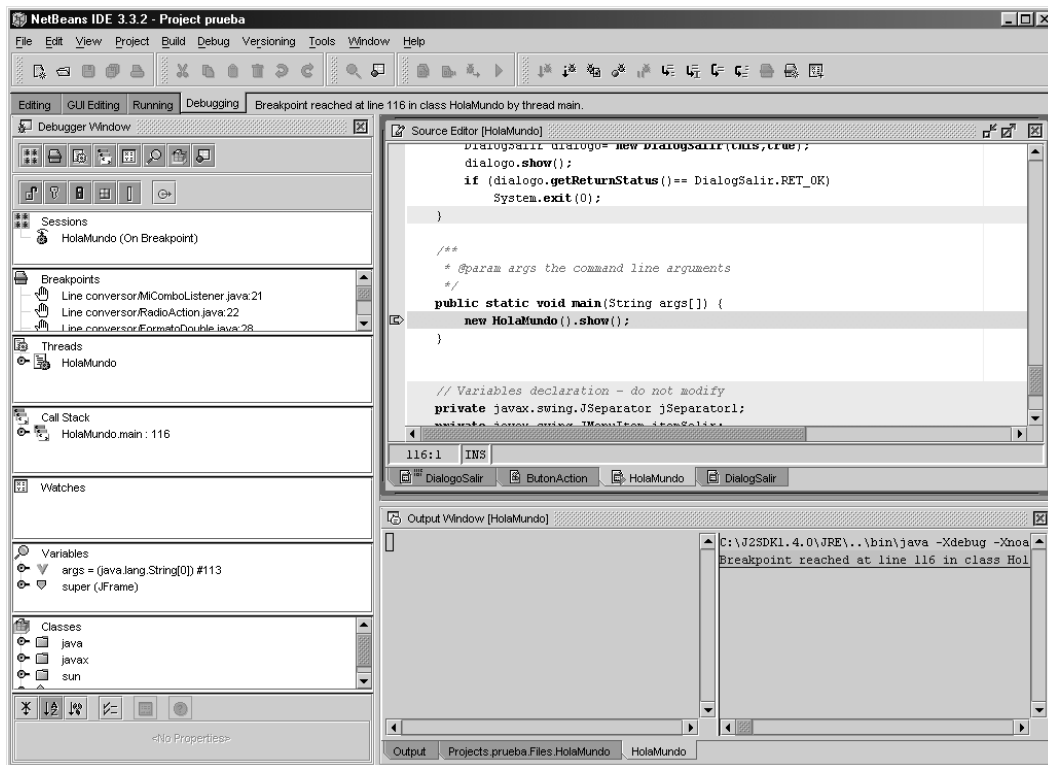
/**
 * @param args the command line arguments
 */
public static void main(String args[]) {
 new HolaMundo().show();
}

// Variables declaration - do not modify
private javax.swing.JSeparator jSeparator1;
private javax.swing.JMenuItem itemSalir;
private javax.swing.JMenuItem itemAyuda;
private javax.swing.JPopupMenu popUp;
private javax.swing.JToggleButton jToggleButton1;
// End of variables declaration
}

9:21  INS
ButonAction DialogoSalir HolaMundo DialogSalir
```

Para iniciar una ejecución de este estilo se debe seleccionar *Debug|Start* y la ejecución del programa se parará cuando llega a la línea donde se puso el breakpoint.

El *workspace* de depuración incluye tres ventanas: la ventana de depuración ( *Debugger*), la de Salida ( *Output*), y la del editor del código.



La ventana de depuración contiene siete vistas de lo que esta ocurriendo en el programa: las sesiones (lista los programas que se están depurando), los breakpoints, los threads, la pila de llamadas, los watches, las variables, y las clases.

En estos momentos se pueden comprobar el valor de todos los atributos y variables que son accesibles desde ese punto. Para continuar con la ejecución, se puede realizar de diferentes maneras:

- Ejecución paso a paso. Si se selecciona *Debug|Step into* o mejor F7, se ejecutará la línea de código. Si esa línea es una llamada a un método, entrará a dicho método. Con la opción *Debug|Step over* o F8 también ejecuta la línea, pero en caso de ser una llamada a un método, ejecuta dicho método sin entrar dentro.
- Continuar. Si seleccionamos *Debug|Continue* devuelve el control al programa y sólo volverá a pararse si existe otro breakpoint.



En cualquier momento se puede dar por finalizado este tipo de ejecución seleccionando *Debug|Finish*.

## 6. Direcciones Web

- Página principal de NetBeans:

<http://www.netbeans.org/>

## 7. Soluciones a las Actividades y Ejercicios

### 7.1. Actividad Propuesta 1

Para que aparezca el Menú contextual, debemos añadir un *MouseListener* al Frame. La construcción del menú se hace mediante el diseñador de NetBeans.

El código final resultante es el siguiente:

```
/*
 * HolaMundo.java
 *
 * Created on      8 de julio de 2002, 13:29
 */

package es.uc3m.inf.dei.fse02.msicilia;
import javax.swing.WindowConstants;

/**
 *
 * @author Miguel Ángel Sicilia
 */
public class HolaMundo extends javax.swing.JFrame {

    /** Creates new form HolaMundo */
    public HolaMundo() {
        System.out.println("Hola, Mundo");
        initComponents();
        setDefaultCloseOperation(WindowConstants.DO_NOTHING_ON_CLOSE);
    }
}
```



```
// pop-up:
addMouseListener(new java.awt.event.MouseAdapter() {
    public void mousePressed(java.awt.event.MouseEvent e) {
        maybeShowPopup(e);
    }

    public void mouseReleased(java.awt.event.MouseEvent e) {
        maybeShowPopup(e);
    }

    private void maybeShowPopup(java.awt.event.MouseEvent e) {
        if (e.isPopupTrigger()) {
            jPopupMenu1.show(e.getComponent(),
                e.getX(), e.getY());
        }
    }
});

}

/** This method is called from within the constructor to
 * initialize the form.
 * WARNING: Do NOT modify this code. The content of this method is
 * always regenerated by the Form Editor.
 */
private void initComponents() {
    jPopupMenu1 = new javax.swing.JPopupMenu();
    jMenuItem1 = new javax.swing.JMenuItem();
    jSeparator1 = new javax.swing.JSeparator();
    jMenuItem2 = new javax.swing.JMenuItem();
    jToggleButton1 = new javax.swing.JToggleButton();

    jMenuItem1.setText("Salir");
    jMenuItem1.addMouseListener(new java.awt.event.MouseAdapter() {
        public void mousePressed(java.awt.event.MouseEvent evt) {
            jMenuItem1MousePressed(evt);
        }
    });

    jPopupMenu1.add(jMenuItem1);
    jPopupMenu1.add(jSeparator1);
    jMenuItem2.setText("Ayuda");
    jMenuItem2.addMouseListener(new java.awt.event.MouseAdapter() {
        public void mousePressed(java.awt.event.MouseEvent evt) {
            jMenuItem2MousePressed(evt);
        }
    });
});
```



```
jPopupMenu1.add(jMenuItem2);

getContentPane().setLayout(new java.awt.FlowLayout());

setTitle("Mi Frame");
setBackground(new java.awt.Color(153, 255, 0));
addWindowListener(new java.awt.event.WindowAdapter() {
    public void windowClosing(java.awt.event.WindowEvent evt) {
        exitForm(evt);
    }
});

jToggleButton1.setText("Mi Boton");
jToggleButton1.addMouseListener(new java.awt.event.MouseAdapter() {
    public void mouseClicked(java.awt.event.MouseEvent evt) {
        jToggleButton1MouseClicked(evt);
    }
});

getContentPane().add(jToggleButton1);

pack();
}

private void jMenuItem2MousePressed(java.awt.event.MouseEvent evt) {
    // Add your handling code here:
    javax.swing.JOptionPane.showMessageDialog(evt.getComponent(),
        "Aplicacion escrita por Miguel Angel");
}

private void jMenuItem1MousePressed(java.awt.event.MouseEvent evt) {
    // Add your handling code here:
    // HolaMundo h = (HolaMundo)evt.getComponent();
    exitForm(null);
}

private void jToggleButton1MouseClicked(java.awt.event.MouseEvent evt)
{
    // Add your handling code here:
    if (jToggleButton1.isSelected())
        jToggleButton1.setText("Pulsado");
    else
        jToggleButton1.setText("No Pulsado");
}

/** Exit the Application */
```



```
private void exitForm(java.awt.event.WindowEvent evt) {
    DialogoSalir d = new DialogoSalir(this, true);
    d.show();
    if (d.getReturnStatus()==DialogoSalir.RET_OK)
        System.exit(0);
}

/**
 * @param args the command line arguments
 */
public static void main(String args[]) {
    new HolaMundo().show();
}

// Variables declaration - do not modify
private javax.swing.JToggleButton jToggleButton1;
private javax.swing.JMenuItem jMenuItem2;
private javax.swing.JMenuItem jMenuItem1;
private javax.swing.JSeparator jSeparator1;
private javax.swing.JPopupMenu jPopupMenu1;
// End of variables declaration
}
```