



Introducción

En este laboratorio desarrollaremos un caso de uso que verifica si una cuenta y un password son correctos. En dicho caso de uso la presentación y la lógica del negocio se definirán en dos capas diferentes, y la presentación accederá a la lógica del negocio por medio de una interfaz Java.

Objetivos

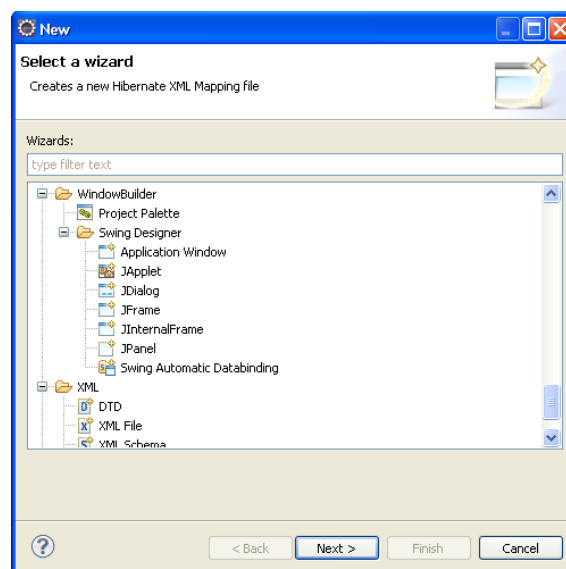
Los objetivos del laboratorio son los siguientes:

- Definir interfaces gráficas de usuario usando la API Swing de Java, y utilizando algunos componentes básicos como cajas de texto, botones, botones de opción, listas desplegables y listas.
- Entender cómo se pueden separar las capas de presentación y lógica del negocio por medio de interfaces, los cuales permiten una independencia de la presentación y la lógica del negocio. La presentación no conocerá exactamente el nombre de la clase con la lógica del negocio que utiliza.

Pasos a seguir

0. Comprobar la correcta instalación de WindowBuilder

Si al seleccionar File => New => Other vemos que disponemos del asistente WindowBuilder con el Swing Designer entonces no hay que hacer nada más, porque ya está instalado.




En otro caso, tendríamos que instalarlo. Para ello se puede descargar y descomprimir el zip correspondiente a la distribución del Eclipse con la que estemos trabajando disponible en: <http://www.eclipse.org/windowbuilder/download.php>.



En dicha página se encuentran disponibles los enlaces a los sitios de actualización. Se puede hacer de tres maneras: 1) usando el Eclipse Marketplace 2) descargando el código de WindowBuilder (Zipped Update Site) y descomprimirlo después; o 3) obteniendo la dirección (Update Site) a partir de la cual Eclipse instalará WindowBuilder (Click Dcho sobre el “link” del Update Site => Copiar la ruta del enlace)

Update Sites

Version	Download and Install		
	Update Site	Zipped Update Site	Marketplace
Latest (1.9.4)	link	link	 Install
Last Good Build	link	link	 Install
1.9.4 (Permanent)	link	link	Drag to Install! Drag to your running Eclipse* workspace. *Requires Eclipse Marketplace Client
1.9.3 (Permanent)	link	link	

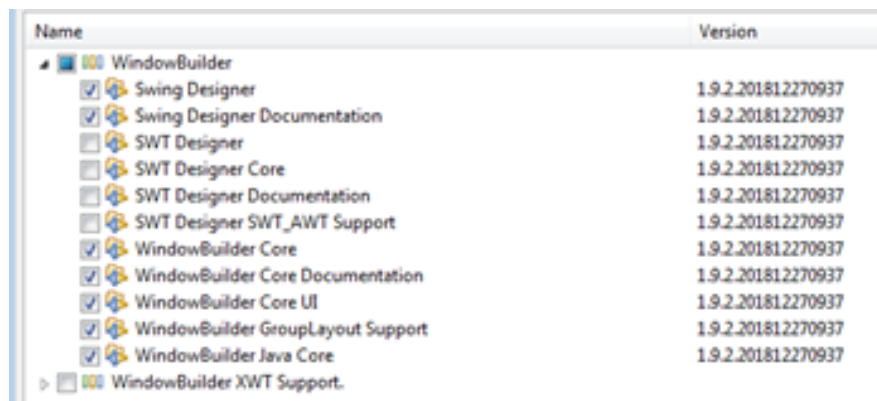
1) Para instalarlo con Eclipse Marketplace, hay que hacer en Eclipse: Help => Eclipse Market Place

=> Sobre la ventana (cliente de Eclipse Marketplace) que aparece se puede “arrastrar” el icono de “Install”, o bien se puede buscar “Window Builder” => Instalar => Confirm

Las otras dos opciones lo instalan desde el “Update Site”. Para ello en Eclipse se hace: Help => Install New Software => Add =>

- 2) Si hemos descargado el código, entonces:
Local => Seleccionar el directorio donde hayamos descomprimido el WindowBuilder
- 3) Y si hemos obtenido la dirección de actualización, entonces:
Location => Introducir la dirección de actualización

A continuación activar las opciones de WindowBuilder Engine y Swing Designer => Next



Tras la instalación, nos pedirá reiniciar el eclipse.



1. Crear el proyecto Java

File => New => Project => Java Project => Dar un nombre al proyecto, como por ejemplo Lab1

Para desarrollar la aplicación crearemos 4 prototipos:

A. Primer prototipo

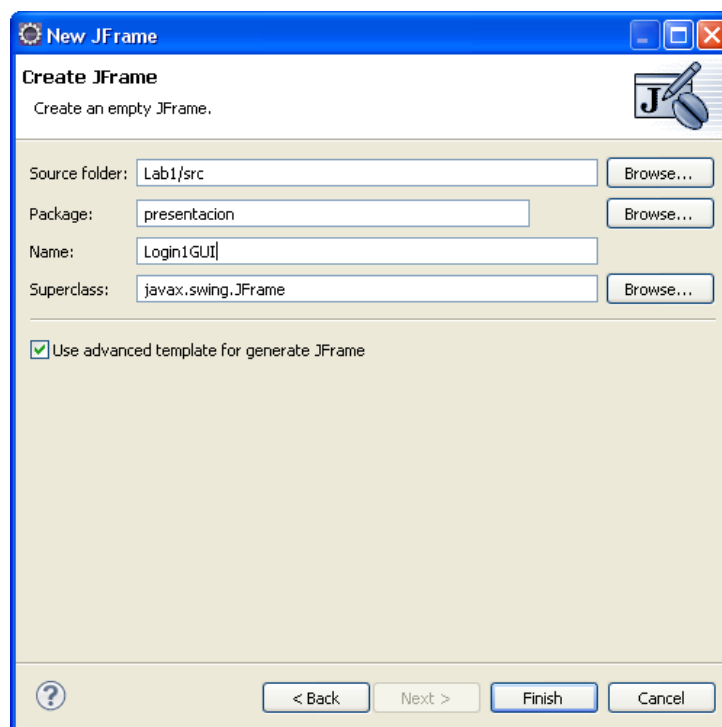
En este primer prototipo definiremos 2 clases. Una clase será la interfaz gráfica (capa de presentación) y la otra será la lógica del negocio, tal y como aparecen en la siguiente figura:



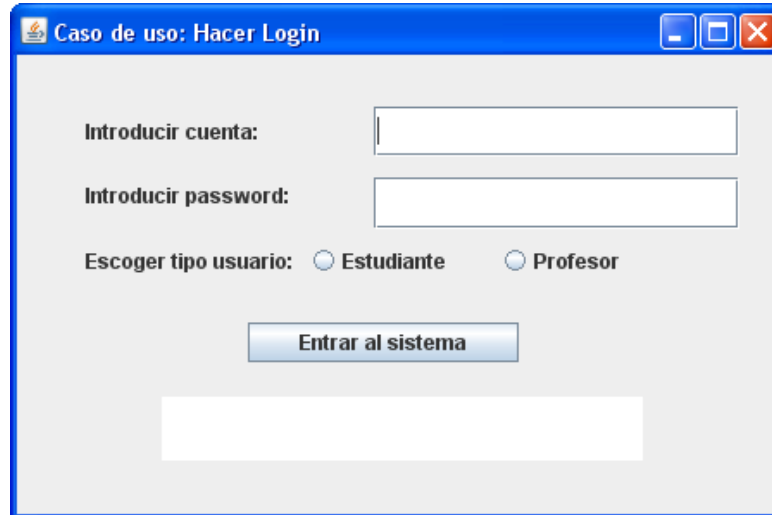
Para ello, seguir los siguientes pasos.

A.1) Crear la presentación (interfaz gráfica de usuario o GUI)

Crear un nuevo Frame (clase Login1GUI), con el siguiente formato. Para crear el frame, escoged File => New => Other => WindowBuilder => Swing Designer => JFrame => Next => Poner presentacion como nombre del paquete y Login1GUI como nombre de la clase, tal y como aparece a continuación; y terminar con "Finish"



Al posicionarse encima de la clase Login1GUI vemos que se puede ver el código de la clase en la pestaña “Source” como el diseño gráfico en la pestaña “Design”. Usaremos la vista “Design” para crear de manera visual la siguiente ventana (o JFrame):



Para ello hay que poner título a la ventana. Seleccionar el JFrame en la vista “Components” y escribir el texto “Caso de uso: Hacer Login” en la propiedad “title” de la vista “properties”. El resto de componentes gráficos 3 etiquetas (JLabel), 2 cajas de texto (uno JTextField al que llamaremos `textField` para que no dé problemas el código que usaremos después y el otro, JPasswordField, al que llamaremos `passwordField`), 1 botón (JButton), 2 botones de opción (JRadioButton, a los que llamaremos `rdbtnProfesor` y `rdbtnEstudiante`) y un área de texto (JTextArea a la que llamaremos `textArea`) se diseñan de una manera muy sencilla: Hacer click en el icono correspondiente (JLabel, JButton, ...) de la vista “Palette” y colocarlo en el lugar deseado del JFrame que estamos diseñando, haciendo click de nuevo y escribiendo el nombre de la etiqueta que se desee. Se puede mover y redimensionar utilizando el ratón.

Nota: si no pudierais poner los componentes gráficos en el lugar que quisierais dentro del JFrame, entonces, es que tiene asignado un administrador de diseño (o layout). Para quitarlo: hay que seleccionar el objeto “contentPane” en la vista “Components”, y a continuación seleccionar el layout “Absolute” en la propiedad “Layout” de la vista “properties”.

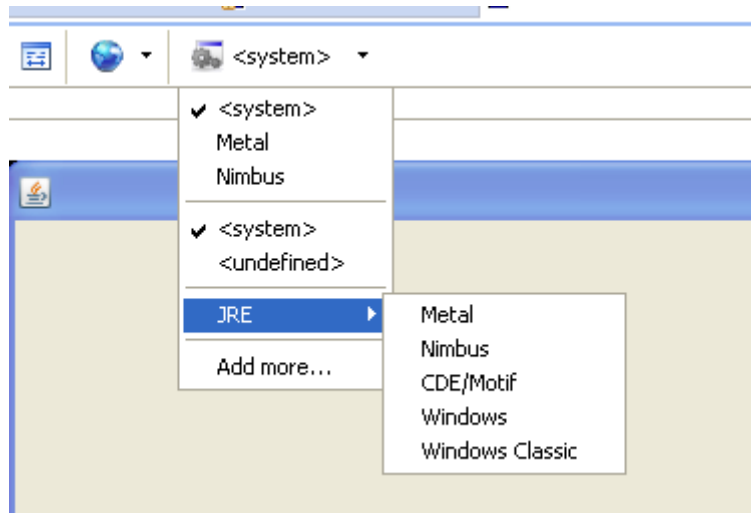
También pueden visualizarse y cambiarse los valores de las propiedades de todos estos objetos gráficos (que se pueden ver y seleccionar en la vista “Components”)

Se podría ver el resultado obtenido hasta el momento si ejecutamos dicha clase. Para ello, hay que posicionarse en la vista “Package explorer” sobre la clase LoginGUI=> Hacer Click derecho => Run As => Java Application

Nota: si la visualización de la ventana en ejecución no coincide con la que aparece en el diseño realizado con WindowBuilder, podéis forzar a que lo sea seleccionando lo siguiente:

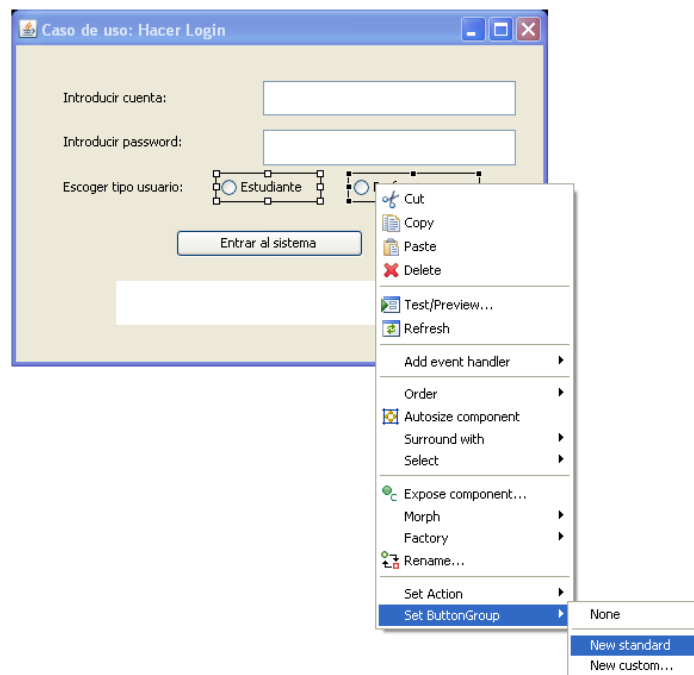
Window => Preferences => WindowBuilder => Swing => LookAndFeel => Activar la opción: Apply chosen LookAndFeel in main() method.

Podéis cambiar el look-n-feel de las ventanas seleccionando alguno de los disponibles en la pestaña “Select look-n-feel” del WindowBuilder:



Probad a escribir nombres en las cajas de texto y comprobad que en el correspondiente al password no se muestran los caracteres tecleados. Se puede comprobar que permite seleccionar el tipo de usuario “Estudiante” y “Profesor” a la vez. Tampoco sucede nada si pulsamos el botón “Entrar al sistema”...

De momento, arreglaremos solamente el problema de que los 2 botones de opción no sean excluyentes. Para ello, se seleccionan ambos JRadioButton (primero uno y después manteniendo la tecla de mayúsculas el otro), y haciendo click derecho se crea un nuevo grupo de botones para ambos (Set ButtonGroup => New standard)





Esto simplemente, genera el código siguiente de creación de un objeto de ButtonGroup, al cual se le añaden los 2 objetos JRadioButton:

```
private final ButtonGroup buttonGroup = new ButtonGroup();  
buttonGroup.add(rdbtnEstudiante);  
buttonGroup.add(rdbtnProfesor);
```

A estas alturas conviene recordar que este código Java (y todo el anterior) se podía haber escrito de manera manual, pero que es evidente que utilizar la herramienta WindowBuilder para generarlo de manera automática es más productivo.

CONCEPTO IMPORTANTE: En Swing, todos los componentes gráficos (botones, cajas de texto, opciones radiales, etc. se modelan como objetos de clases Java, los cuales se crean (con “new”) y se les pide ejecutar acciones y actualizar propiedades (métodos “set”, “get”, ...). Algunos de esos componentes son contenedores como los paneles (JPanel) y las ventanas (JFrame) a los cuales se les añaden componentes (métodos “add”), que pueden ser a su vez contenedores, y forman una jerarquía de objetos gráficos.

La definición y uso de los objetos se puede ver en la pestaña “Source” y la jerarquía de objetos en la vista “Components” de la pestaña “Design”

A.2) Crear la lógica del negocio

Definir la clase Login1, que implementa la lógica del negocio, esto es, que comprueba si la cuenta y el password son correctos. Esta clase la crearemos dentro del paquete logicaNegocio. En la clase que presentamos a continuación, la cuenta y el password son correctos cuando ambos son iguales.

Para ello, hacer File => New => Class => Escribir “logicaNegocio” en package y Login1 en “name” y escribir el método “hacerLogin” siguiente

```
package logicaNegocio;  
  
public class Login1 {  
    public boolean hacerLogin(String login, String password, String tipo){  
        return login.compareTo(password)==0;  
    }  
}
```

A.3) Conectar la presentación con la lógica del negocio

Para ello, hay que incrustar en el objeto de presentación el objeto con la lógica del negocio e invocar a la lógica del negocio cuando corresponda, esto es, cuando suceda el evento correspondiente.

A.3.1) Incrustar la lógica del negocio en la presentación

Para incrustar el objeto con la lógica del negocio definiremos un atributo en la clase que permitirá almacenarlo y un método con el cual se podrá asignar cuando se desee.

El atributo se debe añadir escribiendo el código siguiente en la pestaña “Source” de la clase “LoginGUI”

```
private Login1 logicaNegocio;
```

Sin olvidar añadir la instrucción: `import logicaNegocio.Login1;`

Para asignar valores al atributo anterior, definiremos también el método `setLogicaNegocio` en la clase de presentación `LoginGUI`

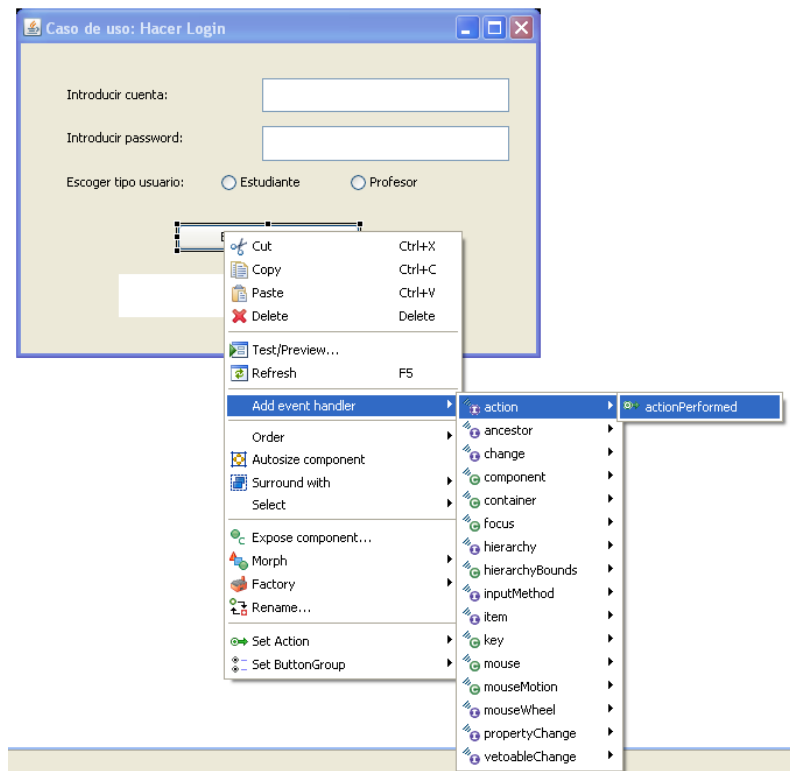
```
public void setLogicaNegocio(Login1 nl) {
    logicaNegocio=nl;
}
```

A.3.2) Invocar a la lógica del negocio cuando suceda el evento deseado

En este punto, vamos a solucionar el problema pendiente del paso A.1), esto es, que no había respuesta cuando se pulsaba el botón “Entrar al sistema”. Lo que haremos es que al pulsar el botón se invoque a la lógica del negocio encargada de decidir si se puede o no hacer login.

Para ello hay que programar un método que se ejecutará cuando, en tiempo de ejecución, suceda un evento determinado (pulsar el botón en este caso), para lo cual hace falta saber cómo se llama ese método y dónde se encuentra. Si hacemos click derecho sobre el botón etiquetado con “Entrar al sistema” => Add event handler => Action => Action Performed nos situará exactamente en el método a programar, el `actionPerformed`.

Truco: bastaría simplemente con haber hecho doble click sobre el botón.



El código que se generará de manera automática será el siguiente:



```
JButton btnEntrarAlSistema = new JButton("Entrar al sistema");  
btnEntrarAlSistema.addActionListener(new ActionListener() {  
    public void actionPerformed(ActionEvent e) {  
    }  
});
```

A continuación habrá que programar el método `actionPerformed` para que invoque a la lógica del negocio con los valores de la cuenta de usuario, el password y el tipo de usuario almacenados en los componentes gráficos. Dependiendo del resultado, se escribirá un mensaje de texto en el área de texto.

Se obtiene el valor actual en la caja de texto que contiene la cuenta de usuario

```
String l=textField.getText();
```

Se obtiene el valor actual en la caja de texto que contiene el password. Aunque podría hacerse utilizando el método `getText()`, dicho método se encuentra “desaprobado” (`deprecated`) para el componente y aconsejan utilizar `getPassword` en su lugar.

```
String p= new String(passwordField.getPassword());
```

Se obtiene el texto del botón de opción (`JRadioButton`) que se encuentre activo. Una manera de hacerlo, ya que solo hay dos opciones¹, podría ser preguntar a ambas si están o no seleccionadas:

```
String u;  
if (rdbtnProfesor.isSelected())  
    u=rdbtnProfesor.getText();  
else  
    if (rdbtnEstudiante.isSelected())  
        u=rdbtnEstudiante.getText();  
    else u="";
```

Se invoca a la lógica del negocio:

```
boolean b=logicaNegocio.hacerLogin(l,p,u);
```

Y se escribe en el área de texto un mensaje acorde al resultado obtenido

```
if (b) textArea.setText("OK");  
else textArea.setText("Error");
```

¹ Nota: Lo mismo podría hacerse pidiendo al `ButtonGroup` cuál es el botón activo

```
String u=buttonGroup.getSelection().getActionCommand();
```

Pero eso sólo funciona si explícitamente a cada botón de opción se le asigna previamente una acción de manera explícita:

```
rdbtnEstudiante.setActionCommand("Estudiante");  
rdbtnProfesor.setActionCommand("Profesor");
```




El código del método sería el siguiente:

```
btnEntrarAlSistema.addActionListener(new ActionListener() {  
    public void actionPerformed(ActionEvent e) {  
        System.out.println("Evento producido sobre componente: "  
            +e.getActionCommand());  
        String l=textField.getText();  
        String p= new String(passwordField.getPassword());  
        String u;  
        if (rdbtnProfesor.isSelected())  
            u=rdbtnProfesor.getText();  
        else  
            if (rdbtnEstudiante.isSelected())  
                u=rdbtnEstudiante.getText();  
            else u="";  
        boolean b=logicaNegocio.hacerLogin(l, p,u);  
  
        if (b) textArea.setText("OK");  
        else textArea.setText("Error")  
    }  
});
```

¿Hay ERRORES? Puede ser que tengáis algún error con los identificadores `textField`, `passwordField`, `rdbtnProfesor`, `rdbtnEstudiante` o `textArea`. Aseguraros de que sean ATRIBUTOS de la clase y no variables definidas en el método constructor `public LoginGUI()`.

Se puede hacer a mano, o utilizando las facilidades de REFACTORIZACIÓN de código de Eclipse. Para ello: seleccionar haciendo click la variable en el código => Hacer click derecho => Refactor => Convert Local Variable to Field

Igual habéis intentado ejecutar la clase `LoginGUI` y obtenéis un `NullPointerException`, porque todavía no se ha asignado la lógica del negocio en el atributo `logicaNegocio`. Se realiza en el siguiente apartado.

```
public class Login1GUI extends JFrame {  
  
    // Declaración de los atributos de la clase  
    private JTextField textField;  
    private JPasswordField passwordField;
```

CONCEPTO IMPORTANTE: En Swing, ya hemos dicho que los componentes gráficos son OBJETOS, pero también lo son tanto los eventos como los manejadores de eventos (o listener o handlers) que son los objetos de clases que implementan las respuestas a los eventos. En el código anterior puede verse que al objeto botón (almacenado en `btnEntrarAlSistema`) se le añade un objeto manejador/listener/handler de una clase anónima (no se le da ningún nombre a la clase y se define ahí mismo). Por último, en la implementación del método de respuesta se puede hacer uso de información de contexto del evento, que se pasa como parámetro, el `ActionEvent e`.

En el código proporcionado se ve que se escribe en la consola información de contexto sobre el evento generado, en concreto, el texto del componente gráfico sobre el que ha ocurrido el evento (llamando a: `e.getActionCommand()`)



A.4) Lanzar la aplicación: ejecutar la presentación con la lógica del negocio

Aunque podría hacerse en el método principal de la clase de presentación, podemos crear una nueva clase Lanzador que cree el objeto de presentación, le asigne la lógica del negocio y muestre o haga visible la ventana.

```
import logicaNegocio.*;
import presentacion.*;

public class Lanzador {

    public static void main(String[] args) {
        Login1GUI a=new Login1GUI();
        Login1 nl=new Login1();
        a.setLogicaNegocio(nl);
        a.setVisible(true);
    }
}
```

Probad a ejecutarla y comprobad que al pulsar en el botón escribe OK cuando el nombre de cuenta y el password coinciden y Error en otro caso.

El primer prototipo ya está terminado, pero ahora nos gustaría hacerlo más extensible.

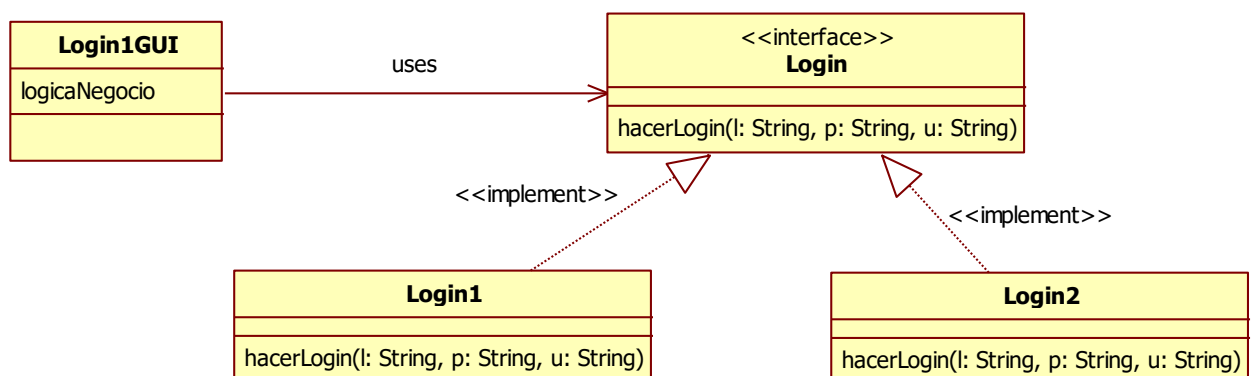
¿Qué habría que hacer si quisiéramos que en la clase de presentación se ejecutara otra lógica de negocio, esto es, si quisiéramos que ejecutara la lógica definida en otra clase Login2, en vez de Login1.

Nos gustaría que se pudiera cambiar la lógica del negocio, **sin que hubiera que cambiar absolutamente nada en el código de la clase de presentación.**

SOLUCIÓN: usando interfaces Java

B. Segundo prototipo

En este segundo prototipo, la presentación y la lógica del negocio se definirán utilizando interfaces Java, tal y como aparece en la siguiente figura:





Una interfaz Java es una clase especial Java que define la especificación de algunos métodos. En la misma, sólo aparece la signatura de los métodos, no se proporciona la implementación a ningún método. A continuación mostramos el código de la interfaz Login. Para crear una interfaz: File => New => Interface => Dar “logicaNegocio” como nombre del paquete y “Login” como nombre de la interfaz.

```
package logicaNegocio;

public interface Login {
    public boolean hacerLogin(String l, String p, String u);
}
```

A continuación presentamos una clase concreta que implementa la interfaz. Si una clase dice que implementa la interfaz, entonces debe implementar todos los métodos. En realidad, se trata de la redefinición de la clase Login1 anterior, donde añadimos el código que dice que implementa la interfaz Login (**implements** Login):

```
package logicaNegocio;

//public class Login1 {
public class Login1 implements Login{
    public boolean hacerLogin(String login, String password, String tipo){
        return login.compareTo(password)==0;    }
}
```

Es muy importante entender la relación entre la interfaz y la clase. Si una clase implementa la interfaz, los objetos de esa clase son objetos de la clase interfaz, esto es, en nuestro caso es correcta la siguiente instrucción:

```
Login nl=new Login1();
```

Como la clase Login1 implementa la interfaz Login, los objetos de Login1 son también del tipo Login.

A continuación realizaremos algunos cambios en la clase de presentacion del prototipo, para que trabaje también con la interfaz Login.

a) En la clase de presentacion Login1GUI, el atributo que permite almacenar la lógica del negocio será del tipo interfaz Login. Esto quiere decir, que ese atributo puede almacenar objetos de cualquier clase que implemente la interfaz Login.

```
private Login logicaNegocio;
```

b) Redefiniremos el método para asignar la lógica del negocio, para que trabaje con objetos de la interfaz Login, y no de la clase Login1:

```
public void setLogicaNegocio(Login nl){
    logicaNegocio =nl;
}
```



c) Y cambiaremos la clase Lanzador:

```
import logicaNegocio.*;
import presentacion.*;

public class Lanzador {

    public static void main(String[] args) {
        Login1GUI a=new Login1GUI();
        Login nl=new Login1();           // nl es del tipo interfaz Login
        a.setLogicaNegocio(nl);
        a.setVisible(true);
    }
}
```

Hasta ahora, tenemos la misma funcionalidad que habíamos definido para el primer prototipo. Ahora definiremos una nueva lógica del negocio Login2, donde siempre dejará entrar a cualquier estudiante, pero en el caso de los profesores, sólo si el login y el password tienen el mismo número de caracteres. Este es el código:

```
package logicaNegocio;

public class Login2 implements Login{
    public boolean hacerLogin(String l, String p, String u){
        if (u.equals("Estudiante")) return true;
        else return l.length()==p.length(); }
}
```

Para modificar la lógica del negocio asociada a la presentación, sin tener que cambiar esta última, basta con que se defina esa nueva lógica del negocio en el programa lanzador. Como ambas lógicas del negocio implementan la interfaz Login, no hay ningún problema.

```
import logicaNegocio.*;
import presentacion.*;

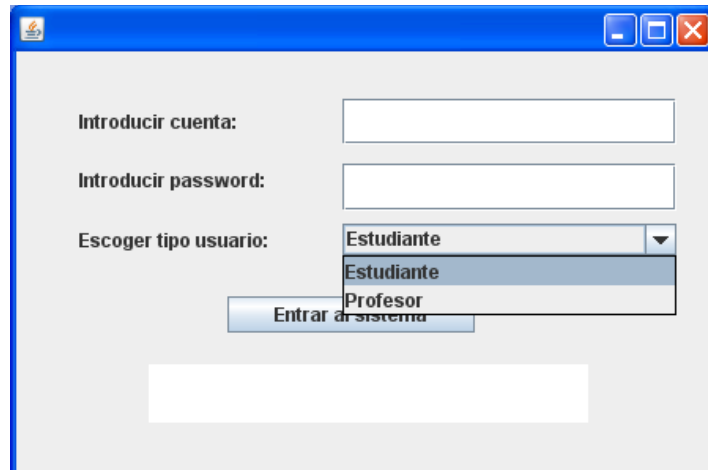
public class Lanzador {

    public static void main(String[] args) {
        Login1GUI a=new Login1GUI();
        Login nl=new Login2();           // Se usa Login2 en vez de Login1
        a.setLogicaNegocio(nl);
        a.setVisible(true);
    }
}
```

Ejecutar la aplicación y comprobar su funcionamiento.

3. Tercer prototipo

En este prototipo y en el siguiente, vamos a cambiar la clase de presentación para utilizar otros componentes gráficos. En vez de utilizar 2 botones de opción para seleccionar el tipo de usuario utilizaremos una lista desplegable.



En este punto, crearemos una nueva clase de presentación que sea una copia de la anterior. Una manera sencilla de hacerlo sería la siguiente:

Crear una nueva clase llamada Login2GUI. File => New => Class => Dar el nombre de clase "Login2GUI" en el paquete "presentacion"

Seleccionar y copiar todo el código de la clase Login1GUI, y pegarlo en la clase Login2GUI (sin olvidar todos los "import" que tal vez tengáis que expandir). En ese momento Eclipse detectará algunos problemas ya que la clase deberá llamarse Login2GUI en vez de Login1GUI en el código fuente. Eclipse os mostrará ese error y os propondrá renombrarla por Login2GUI. Si no, se puede modificar a mano en los 3 lugares donde aparece, o bien, usar las facilidades de refactorización de Eclipse: Seleccionar el nombre LoginGUI => Click derecho => Refactor => Rename => Dar el nombre Login2GUI y enter.

En la pestaña "Design" de la nueva clase Login2GUI (si no aparece se puede mostrar haciendo click sobre Login2GUI en la vista "Package Explorer" => Click Dcho. => Open With => WindowBuilder Editor)

Para sustituir los botones de opción por la lista desplegable hay que:

- Eliminar los dos botones de opción (haciendo click sobre ellos y delete)
- Seleccionar una lista desplegable (un JComboBox en la vista "Palette") y colocarlo en el JFrame a la derecha de la etiqueta "Escoger tipo usuario:".
- Asegurarnos de que la variable del JComboBox se define como un atributo de la clase y no como variable local del método constructor.
- Añadir las opciones a mostrar en la lista desplegable. Para ello hay que:



Añadir un modelo a la lista desplegable. Ese modelo simplemente contendrá los elementos a mostrar en el componente gráfico. En el código fuente de la clase añadiremos un atributo con el modelo:

```
private DefaultComboBoxModel usuarios = new DefaultComboBoxModel();
```

Y dentro del código (por ejemplo, después de crear el objeto JComboBox), asociaremos dicho modelo a la lista desplegable y añadiremos las opciones a mostrar “Estudiante” y “Profesor”.

```
comboBox = new JComboBox(); // Creación del JComboBox  
comboBox.setModel(usuarios);  
usuarios.addElement("Estudiante");  
usuarios.addElement("Profesor");
```

De esta manera, en tiempo de ejecución, se podrán ir añadiendo o quitando elementos del modelo.

Nota: se puede añadir un modelo con programación visual (esto es, sin modificar a mano el código fuente) modificando la propiedad “model” del componente `comboBox` y añadiendo los valores que se desee (`Estudiante` y `Profesor`), pero el modelo no podría modificarse en tiempo de ejecución con el código que genera automáticamente.

En el método de atención al evento habrá que sustituir el código que trabajaba con los botones de opción:

```
if (rdbtnProfesor.isSelected())  
    u=rdbtnProfesor.getText();  
else  
    if (rdbtnEstudiante.isSelected())  
        u=rdbtnEstudiante.getText();  
    else u="";
```

Por este otro código, que obtiene el usuario seleccionado en el JComboBox, que se le pregunta al modelo: `u=usuarios.getSelectedItem().toString();`

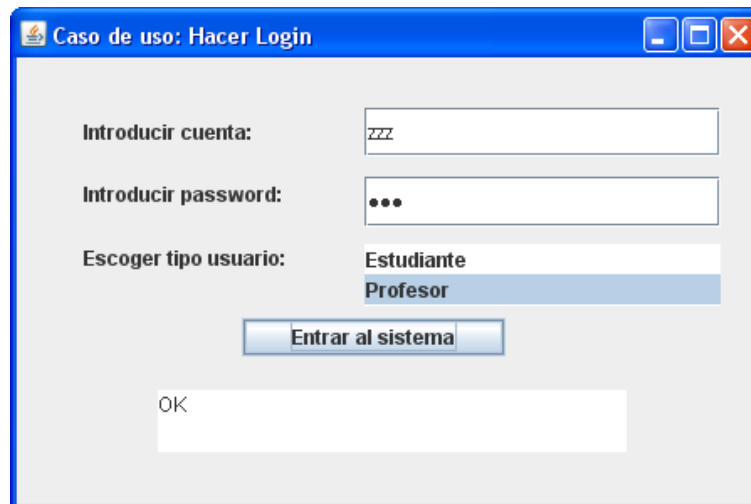
Aunque también podría obtenerse el objeto seleccionado preguntárselo al propio JComboBox: `u=comboBox.getSelectedItem().toString();`

Podemos ejecutar ahora la nueva clase de presentación Login2GUI modificando y ejecutando la clase Lanzador siguiente:

```
import logicaNegocio.*;  
import presentacion.*;  
public class Lanzador {  
    public static void main(String[] args) {  
        Login2GUI a=new Login2GUI(); // Presentación con JComboBox  
        Login nl=new Login2();        // Asignar la lógica del negocio  
        a.setLogicaNegocio(nl);  
        a.setVisible(true);}}}
```

4. Cuarto prototipo

En este último prototipo usaremos otra nueva clase de presentación donde utilizaremos una lista. Para ello crearemos una clase Login3GUI a partir de Login2GUI o de Login1GUI (como en el tercer prototipo) y eliminaremos el JComboBox o los JRadioButton, según sea el caso.



A continuación habría que:

- Seleccionar una lista (un JList en la vista "Palette") y colocarlo en el JFrame a la derecha de la etiqueta "Escoger tipo usuario:".
- Asegurarnos de que la variable del JList se define como un atributo de la clase y no como variable local del método constructor.
- Añadir las opciones a mostrar en la lista. Para ello hay que:

Añadir un modelo a la lista. Ese modelo simplemente contendrá los elementos a mostrar en el componente gráfico. En el código fuente de la clase añadiremos un atributo con el modelo:

```
private DefaultListModel usuarios = new DefaultListModel();
```

Y dentro del código (por ejemplo, después de crear el objeto JList), asociaremos dicho modelo a la lista desplegable y añadiremos las opciones a mostrar "Estudiante" y "Profesor".

```
list = new JList(); // Creación del JList
list.setModel(usuarios);
usuarios.addElement("Estudiante");
usuarios.addElement("Profesor");
```

Nótese: que en tiempo de ejecución se podrán ir añadiendo o quitando elementos del modelo.

En el método de atención al evento habrá que sustituir el código que trabajaba con los botones de opción o con la lista desplegable por el que obtiene el usuario seleccionado en el JList, que se le pregunta al propio JList:



```
u=list.getSelectedValue().toString();
```

Nota: no existe el método `getSelectedItem()` ni en `JList`, ni en `DefaultListModel`, como alguien podría esperar. Lo que hay que hacer es buscar en las APIs de las clases los métodos disponibles y seleccionar el que se necesite. Esto es: tratad de encontrar el método con la ayuda de Google buscando lo siguiente: “API Java `JList`”

Podemos ejecutar ahora la nueva clase de presentación `Login3GUI` modificando y ejecutando la clase `Lanzador` siguiente:

```
import logicaNegocio.*;
import presentacion.*;

public class Lanzador {

    public static void main(String[] args) {
        Login3GUI a=new Login3GUI(); // Presentación con JList
        Login nl=new Login2();        // Asignar la lógica del negocio
        a.setLogicaNegocio(nl);
        a.setVisible(true);
    }
}
```