Introducción

En este laboratorio desarrollaremos una aplicación que verifica si una cuenta y un password son correctos. En dicha aplicación la presentación y la lógica del negocio se definirán en dos capas diferentes.

Objetivos

Los objetivos del laboratorio son los siguientes:

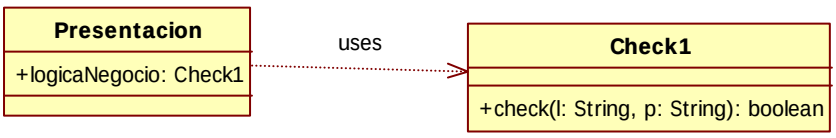
- Entender cómo se pueden separar las capas de presentación y lógica del negocio.
- Mostrar la utilidad de las interfaces Java, las cuales permiten que desde un objeto se pida, en tiempo de ejecución, que se ejecute un método concreto de otro objeto de una clase que **no era conocida en tiempo de compilación** (sí será conocido el nombre del método, que estará definido en la interfaz Java)
- Entender que no es necesario recompilar la capa de presentación cuando se cambia la lógica del negocio, incluso aunque se haga **en tiempo de ejecución**.

Pasos a seguir

Para desarrollar la aplicación crearemos 3 prototipos:

1. Primer prototipo

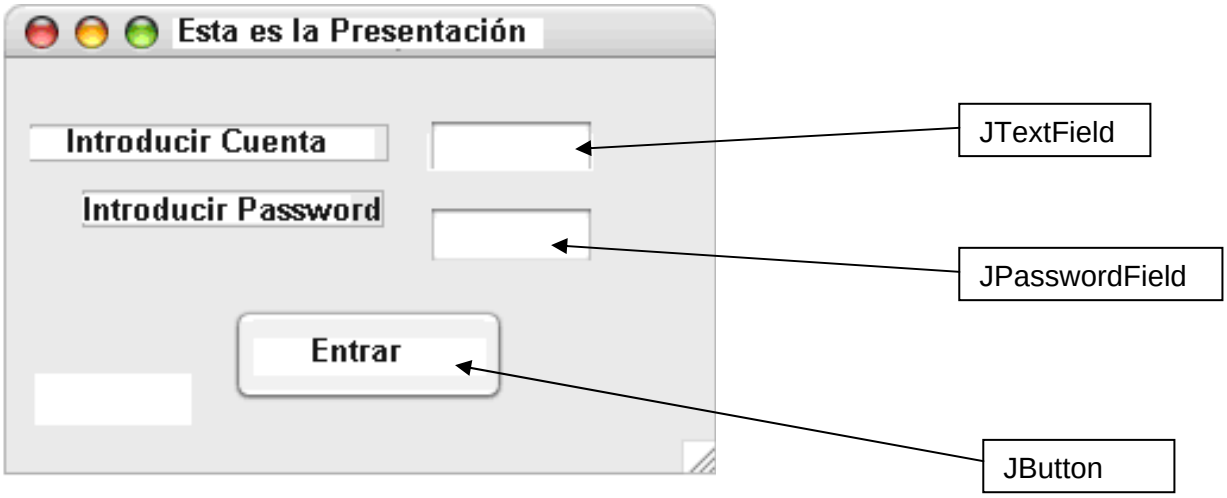
En este primer prototipo definiremos 2 clases. Una clase será la interfaz gráfica (capa de presentación) y la otra será la lógica del negocio, tal y como aparecen en la siguiente figura:



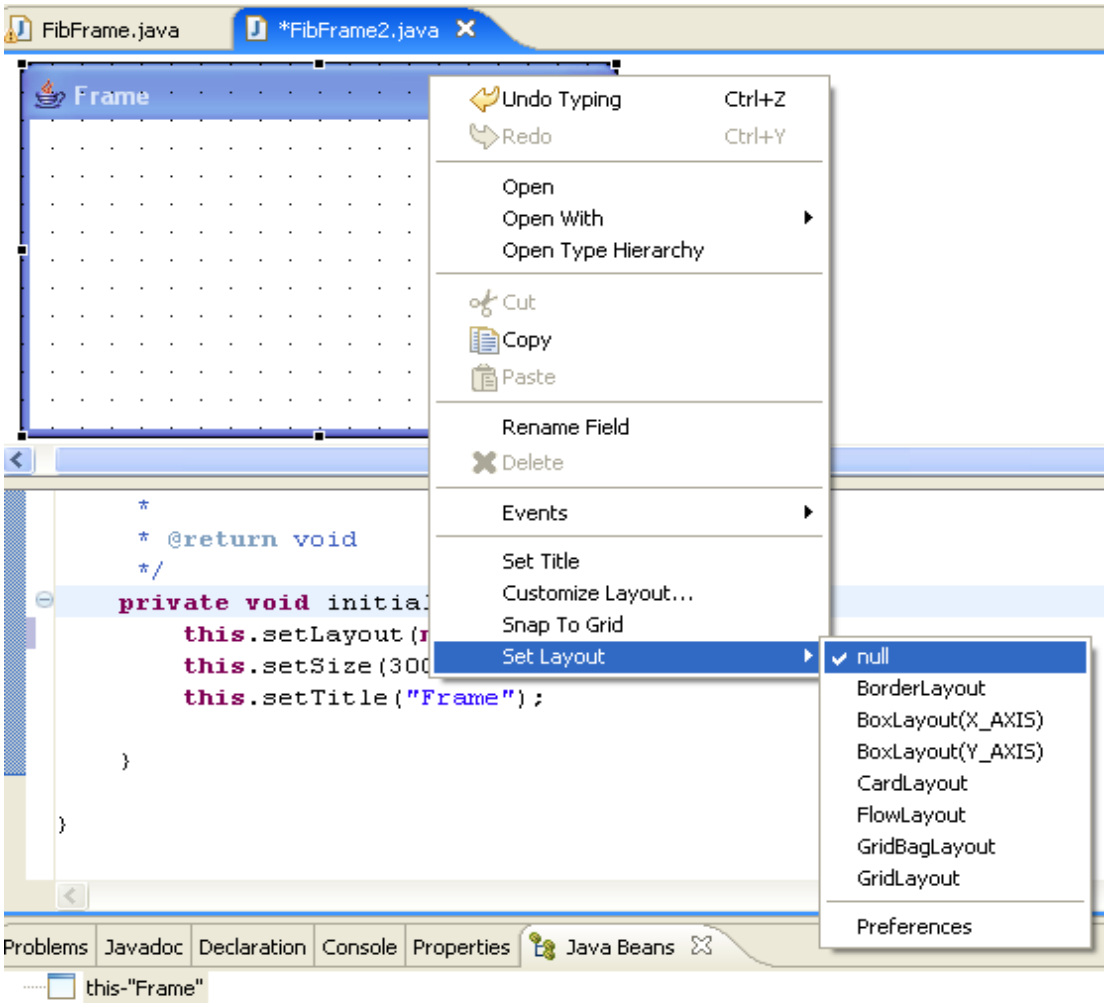
Para ello, seguir los siguientes pasos.

a) Crear un nuevo proyecto Eclipse.
b) Crear un nuevo Frame (clase Presentacion), con el siguiente formato. Para crear el frame, escoged File>>New>>Visual class, y a continuación pulsar la clase Swing>>Frame. Después, rellenad el Frame con dos etiquetas (JLabel), 2 campos de texto (JtextField) y un botón (JButton). También definiremos un área de texto (JTextArea) para devolver el resultado (OK o Error).

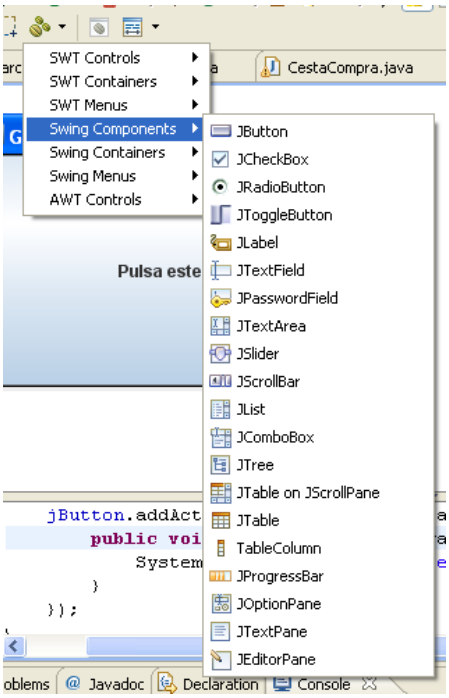
Clase Presentación



Notas:
1) Hay que quitar el administrador de diseño



2) Para poner componentes en el frame, los seleccionaremos de la paleta de componentes (ver siguiente figura) y lo colocaremos en la posición del frame deseada. Para poner el texto asociado al componente, se puede hacer posicionándonos encima del mismo y haciendo click derecho y seleccionando "Set Text".



c) Definir la clase del negocio Check1 que compruebe si la cuenta y el password son correctos. Esta clase la crearemos dentro del paquete logic. En la clase que presentamos a continuación, la cuenta y el password son correctos cuando ambos son iguales.

```
package logic;

public class Check1 {
 boolean check(String login, String password){
       return login.compareTo(password)==0;
 }
}
```

d) En la clase de presentación hay que añadir un atributo de tipo Check1, que sirva para guardar el objeto con la lógica de negocio.

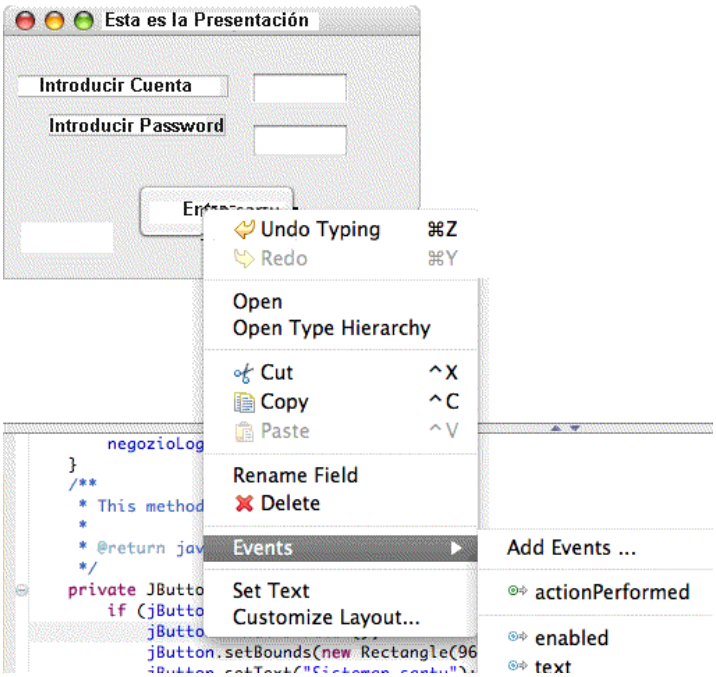
```
     private Check1 logicaNegocio;
```

Es necesario añadir al comienzo la instrucción: `import logic.Check1;`

e) Para asignar valores al atributo anterior, hay que definir el método setLogicaNegocio en la clase de presentación:

```
public void setLogicaNegocio(Check1 nl){
           logicaNegocio=nl;
     }
```

f) Gestionar el evento de pulsar el botón. Para ello hay que colocarse en dicho botón, hacer click en el botón derecho y seleccionar "*actionPerformed*", tal y como aparece en la siguiente figura:



Ese método se ejecutará cada vez que pulsemos el botón.

g) Implementar el método de atención al evento “pulsar botón”. Los datos de entrada se encuentran en los objetos correspondientes a los campos de texto "nameInput" y "passInput".

```
private JButton getJButton() {
      if (jButton == null) {
       jButton = new JButton();
       jButton.setBounds(new Rectangle(96, 107, 115, 41));
       jButton.addActionListener(new
java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent e) {
      String l=nameInput.getText();
      String p=passInput.getText();
      boolean b=logicaNegocio.check(l, p);
      if (b) jTextArea.setText("OK");
        else jTextArea.setText("Error");
    }
  });
}
return jButton;}
```

h) Definir la clase Lanzador, que crea los objetos de presentación y lógica del negocio, y asigna la lógica del negocio a la presentación. Este es el código:

```
public class Lanzador {

      public static void main(String[] args) {
       Presentacion a=new Presentacion();
       Check1 nl=new Check1();
       a.setLogicaNegocio(nl);
       a.setVisible(true);
      }
}
```

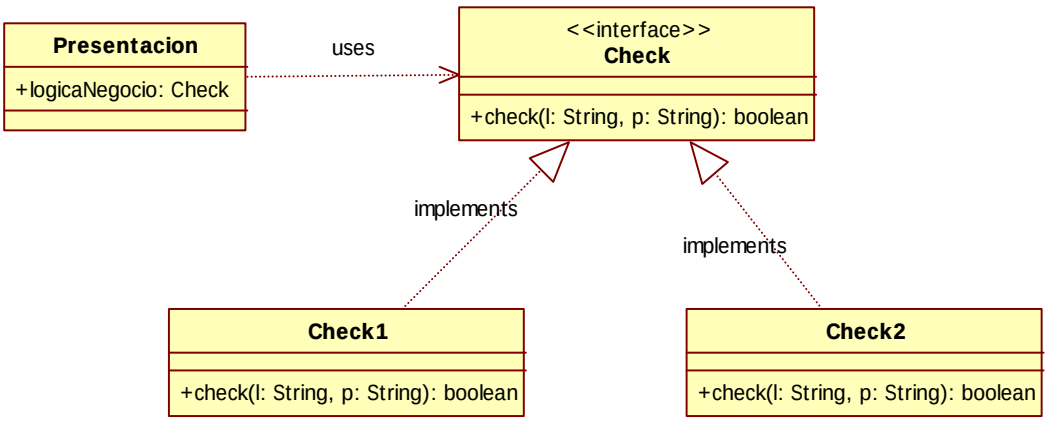
El primer prototipo ya está terminado, pero ahora nos gustaría hacerlo más extensible.

¿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 Check2, en vez de Check1.

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**.

2. 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 Check:

```
package logic;

public interface Check {
 public boolean check(String l, String p);
```

```
}
```

A continuación presentamos una clase concreta que implementa la interfaz. Si una clase dice que implementa la interfaz, entonces debe implementar todos los métodos. Se trata de la redefinición de la clase Check1 que implementa la interfaz Check:

```
package logic;

public class Check1 implements Check{
 public boolean check(String login, String password){
       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:

```
Check nl=new Check1();
```

Como la clase Check1 implementa la interfaz Check, los objetos de Check1 son también del tipo Check.

A continuación realizaremos algunos cambios en la clase Presentacion del prototipo, para que trabaje también con la interfaz Check.

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

```
      private Check logicaNegocio;
```

b) Redefiniremos el método para asignar la lógica del negocio:

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

c) Y cambiaremos la clase Lanzador:

```
public class Lanzador {
  public static void main(String[] args) {
      Presentacion a=new Presentacion ();
      Check nl=new Check1();
      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:

d) Crear una nueva lógica del negocio. En este caso, se confirmará que el login y el password son correctos cuando tienen el mismo número de caracteres. Este es el código:

```
package logic;

public class Check2 implements Check{
       public boolean check(String l, String p){
            return l.length()==p.length();  }
}
```

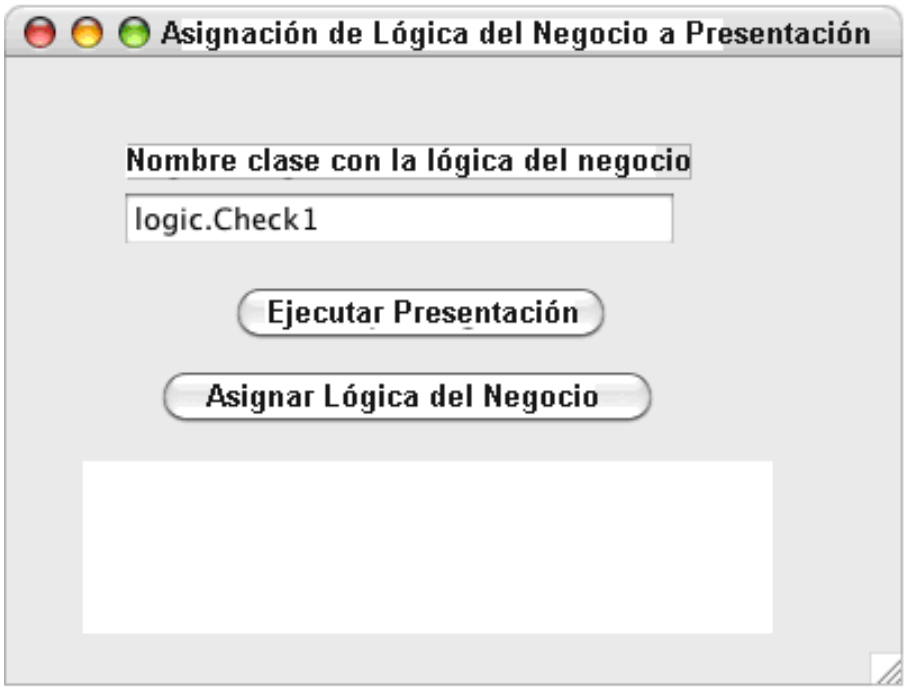
e) Para modificar la lógica del negocio asociada a la presentación, sin tener que cambiar esta última, basta con que se defina esa nueva lógica del negocio en el programa lanzador. Como ambas lógicas del negocio implementan la interfaz Check, no hay ningún problema.

```
public class Lanzador {

  public static void main(String[] args) {
      Presentacion a=new Presentacion();
      Check nl=new Check2();
      a.setLogicaNegocio(nl);
      a.setVisible(true);

     }
}
```

Ejecutar la aplicación y comprobar que funciona.

3. Tercer prototipo

En este último prototipo, definiremos una aplicación que permite cambiar la lógica del negocio en tiempo de ejecución. Para ello, crearemos la clase siguiente: `LanzadorGUI:`



En el primer campo de texto, escribiremos el nombre de la lógica del negocio que queremos asignar a la presentación. Al pulsar el botón etiquetado con "asignar lógica del negocio" se le asignará un objeto de dicha clase cuyo nombre se ha escrito al objeto de presentación. Con el botón "Ejecutar la presentación" se mostrará la ventana de presentación que permite introducir el login y el password.

Estos son los pasos que hay que seguir para desarrollar la aplicación:

a) En la clase LanzadorGUI, hay que crear un objeto de la clase Presentacion y guardarlo como atributo:

```
public class LanzadorGUI extends JFrame {
     private Presentacion presentacion=new Presentacion();
```

b) Definir un método de respuesta al evento "actionPerformed", que se ejecute cuando se pulsa el botón de asignación de la lógica del negocio. El código de dicho método sería el siguiente:

```
private JButton getJButton1() {
 if (jButton1 == null) {
  jButton1 = new JButton();
  jButton1.setBounds(new Rectangle(65, 137, 229, 29));
  jButton1.setText("Asignar              Lógica              Negocio");
jButton1.addActionListener(new java.awt.event.ActionListener() {
 public void actionPerformed(java.awt.event.ActionEvent e) {
  try {
  Check nl = (Check)Class.forName(jTextField.getText()).newInstance();
  presentacion.setLogicaNegocio(nl);
 } catch (Exception e1) {
      e1.printStackTrace();
   }
}
});
}
```

El método Class.forName carga en tiempo de ejecución la clase cuyo nombre coincide con el String que se le pasa como parámetro. A esa clase se le puede enviar el método newInstance que crea un objeto de la misma. El objeto creado, que debe ser del tipo Check si no se quiere que salte una excepción, se le asigna como la lógica del negocio a la presentación (con el método setLogicaNegocio).

c) Al pulsar el botón de “ejecutar la presentación” se ejecutará el código que aparece en el método siguiente:

```
jButton.addActionListener(new java.awt.event.ActionListener() {
  public void actionPerformed(java.awt.event.ActionEvent e) {
    presentacion.setVisible(true);
}
```

d) Definiremos el método main en la clase LanzadorGUI:

```
public static void main(String[] args) {
     LanzadorGUI gui=new LanzadorGUI ();
     gui.setVisible(true);
}
```