Práctica 5. Componentes JavaBeans

Seminario 2. Creación del componente "Luz"

Objetivos

- Conocer los fundamentos de programación de JavaBeans.
- Crear un componente visual JavaBean.
- Aprender las técnicas de comunicación entre componentes.
- Distribuir el componente para incluirlo en un entorno gráfico de programación.

Los JavaBeans son un estándar sobre Java que define el modelo de componentes Sun. Se trata de componentes independientes de la plataforma que pueden ser manipulados visualmente por un Entorno de Desarrollo Integrado (IDE). Para ello es necesario que el bean tenga un interfaz que permita al IDE interrogar al componente en tiempo de diseño (lo que se conoce como mecanismo de introspección).

En esta práctica se creará un componente JavaBean llamado "Luz" que representará un pulsador. El componente tendrá la apariencia de un botón circular con una luz en su parte superior, que puede estar encendida o apagada, para indicar si el pulsador está activo o no. Una propiedad ligada se encargará de notificar a los oyentes (listeners) cuando cambie el estado del pulsador.
La figura 1 muestra la apariencia de este bean.

Figura 1. Apariencia del componente "Luz"

El aspecto del componente se podrá personalizar, cambiando el color de la luz y el tamaño del pulsador. Además se programará de forma gráfica la animación del botón cuando se pulse sobre él haciendo click con el botón izquierdo del ratón.

Primero crearemos un proyecto Java llamado "Componente" y, dentro del proyecto, crearemos un paquete llamado "pulsador" que contendrá los ficheros de nuestro componente.

El fichero principal de nuestro proyecto se llamará "Luz.java"
Nuestro bean heredará de la clase Canvas (java.awt.Canvas) para tener una superficie donde dibujar su representación gráfica. Además implementará el interfaz Serializable para tener la característica de persistencia propia de un bean.

```
package pulsador;

public class Luz extends Canvas implements Serializable {

}
```

Definiendo las propiedades

El bean tendrá tres propiedades:

color: color de la luz del pulsador
encendido: (true/false) – **propiedad ligada** que representa si la luz del pulsador está encendida o apagada
nombre: string para identificar al componente

Además contendrá también los siguientes atributos internos:

encendidoListeners: vector de oyentes interesados en saber que ha cambiado el estado de la propiedad "encendido"
bPulsado: true/false – atributo que indica si el botón de la luz se encuentra pulsado o no. Este atributo determinará si se dibuja el botón presionado o no.

Como todos los beans, el nuestro tendrá un constructor por defecto (sin parámetros)

```
package pulsador;

//import java.awt.Canvas;
import java.awt.*;
import java.io.Serializable;
import java.util.Vector;

public class Luz extends Canvas implements Serializable {
	//propiedades
	private Color color;			//color de la luz
	private boolean encendido=false; //propiedad ligada
	private String nombre;			//identificador del pulsador

	//atributos
	private Vector encendidoListeners = new Vector(); //lista de oyentes
	private boolean bPulsado=false; //indica si el botón está presionado o no

	public Luz() {
	  }
}
```

Para cada una de las propiedades debemos crear un método **set** y un método **get** (hay que considerar también que "encendido" es de tipo boolean).

Debemos tener en cuenta que el método setColor de la propiedad color, cambiará la apariencia gráfica del bean, por lo tanto deberemos realizar una llamada al método repaint() para que se dibuje de nuevo el componente siempre que se cambie el color.

Dibujando el componente

Para dibujar nuestro componente debemos redefinir el método paint() (este método ha sido heredado de Canvas).
Queremos que la apariencia de nuestro componente sea siempre circular, es decir que no se deforme. Para ello debe tener siempre la misma anchura que altura. Esto lo logramos bloqueando su relación de aspecto de tal forma que si en algún momento la altura y la anchura del componente tienen valores distintos, la dimensión menor toma el mismo valor que la dimensión mayor. Como esta operación cambia el tamaño del componente, llamaremos al método invalidate() para que se dibuje de nuevo teniendo en cuenta que la superficie ahora es mayor.

```
public synchronized void paint(Graphics g) {
  //obtener el tamaño del pulsador
     int ancho=getSize().width;
     int alto=getSize().height;

     //bloquear relación de aspecto
     if (ancho!=alto) {
        if (ancho<alto) alto=ancho;
        else ancho=alto;
        setSize(ancho,alto);
      invalidate();
     }

     if (!bPulsado) { //dibujar el botón sin pulsar }
     else { //dibujar el botón pulsado }
  }
```

Podemos encontrar toda la información sobre el diseño gráfico del pulsador en el ANEXO A. Todos los valores relativos a coordenadas, desplazamientos y elementos gráficos dentro del pulsador serán valores relativos a las dos variables "ancho" y "alto".

Definiendo el comportamiento del componente

El siguiente paso es definir la reacción del botón ante un evento de pulsación del ratón.
Para ello modificaremos el constructor y aprovecharemos para fijar el tamaño inicial del componente (tamaño 30x30).

Nos interesa capturar dos tipos de evento de ratón:

1) Cuando se pulsa el botón del ratón
2) Cuando se suelta el botón del ratón

En el primer caso haremos que se ejecute el método LuzPressed() y en el segundo haremos que se ejecute el método LuzReleased(). El propio componente (objeto "this") usará una clase anónima adaptador que será quien se encargue de recibir y tratar estos eventos.

```
public Luz() {
      setSize(30,30); //tamaño inicial por defecto del pulsador
      setMinimumSize(new Dimension(30,30));
      repaint();
      //Añadir eventos de ratón
    this.addMouseListener(new java.awt.event.MouseAdapter() {
        public void mousePressed(MouseEvent e) {
          LuzPressed(e);
        }
        public void mouseReleased(MouseEvent e) {
          LuzReleased(e);
        } });
}
```

Falta ahora definir los métodos LuzPressed y LuzReleased, considerando que el botón no cambiará el estado de la luz cuando se presiona el botón, sino en el momento en que se deja de presionar.

El método LuzPressed(MouseEvent e) debe poner a true el valor del atributo bPulsado y llamar a repaint(), para que se actualice la representación gráfica del botón y se dibuje presionado.
El método LuzReleased(MouseEvent e) es un poco más complejo de programar. Primero deberá comprobar si el botón se encontraba pulsado (atributo bPulsado) (hay que tener en cuenta que alguien puede haber pulsado el ratón fuera de la superficie del componente y arrastrado el puntero hasta él antes de soltarlo).
En caso afirmativo es cuando se llevan a cabo las siguientes acciones:

-Se pone el atributo bPulsado a false
-Se comprueba el valor de la propiedad "encendido".
    Si estaba encendido se hace un setEncendido(false) y si estaba apagado se hace un setEncendido(true)
-Se hace una llamada a repaint() para actualizar el dibujo del componente.

```
public void LuzPressed(MouseEvent e){ //escribir el código};
public void LuzReleased(MouseEvent e){
  if (bPulsado) {
        //escribir el código
  }
};
```

Creando el evento

Antes de programar la propiedad ligada, debemos primero crear el evento que se debe enviar a los oyentes cuando cambie el valor de la propiedad ligada "encendido". El evento se llamará "EncendidoEvent"
Para ello incluimos el fichero "EncendidoEvent.java" dentro del paquete "pulsador".

El evento tiene dos valores el valor antiguo de la propiedad ligada y el valor nuevo de la propiedad ligada. Debemos implementar dos métodos que nos permitan consultar esos valores. También tiene un parámetro que es el objeto que envía el evento.

```
package pulsador;

import java.util.EventObject;

public class EncendidoEvent extends EventObject {
  protected boolean oldEncendido, newEncendido;
  public EncendidoEvent(Object fuente, boolean anterior, boolean nuevo) {
    super(fuente);
    newEncendido=nuevo;
    oldEncendido=anterior;
  }
  public boolean getNewEncendido(){ return newEncendido;}
  public boolean getOldEncendido(){ return oldEncendido;}
}
```

Creando el interfaz

Ahora necesitamos programar el interfaz que deben implementar todos los oyentes que estén interesados en conocer el estado del pulsador.
Para ello creamos el fichero "IEncendidoListener.java"
Nuestro interfaz hereda de EventListener, e incluye un único método que los oyentes deben implementar, llamado "enteradoCambioencendido"

```
package pulsador;

import java.util.EventListener;
import java.util.EventObject;

public interface IEncendidoListener extends EventListener {
  public void enteradoCambioEncendido (EventObject e);
}
```

Programando la propiedad ligada

Una vez que tenemos el evento y la interfaz, podemos programar el código de notificación del cambio de estado de la propiedad ligada "encendido"
Para ello modificamos nuestro fichero "Luz.java"

Incluimos los métodos necesarios para añadir y retirar un oyente del vector de oyentes.

```
public synchronized void addEncendidoListener(IEncendidoListener listener){
        encendidoListeners.addElement(listener);
    }
    public synchronized void removeEncendidoListener(IEncendidoListener listener){
        encendidoListeners.removeElement(listener);
    }
```

El valor de la propiedad ligada cambia en el método setEncendido(), por tanto incluiremos dentro de este método el código para notificar a los oyentes el cambio de valor.

```
   public void setEncendido(boolean newEncendido) {
      boolean oldEncendido=encendido;
      encendido = newEncendido;
        if(oldEncendido!=newEncendido){
            EncendidoEvent event=new EncendidoEvent(this, oldEncendido, newEncendido);
            notificarCambioEncendido(event);
        }
    }
```

En el método se crea el evento y se llama al método notificarCambioEncendido, que se encargará de enviar el evento a cada oyente que se encuentre incluido en el vector.

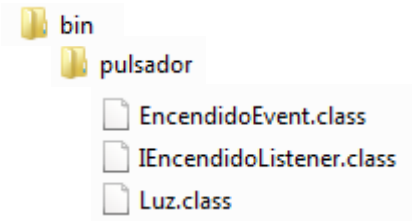
```
    private void notificarCambioEncendido(EncendidoEvent evento){

        //programar el código que envía el evento a los oyentes
      }
```

Añadiendo el código final

```
        public Dimension getPreferredSize() {
                return new Dimension(30, 30);
            }
        //deprecated
        public Dimension getMinimumSize() {
                return getPreferredSize();
            }
```

Empaquetando el bean para su distribución

En la carpeta [eclipse]\workspace\Componente\bin tenemos los ficheros compilados del proyecto.

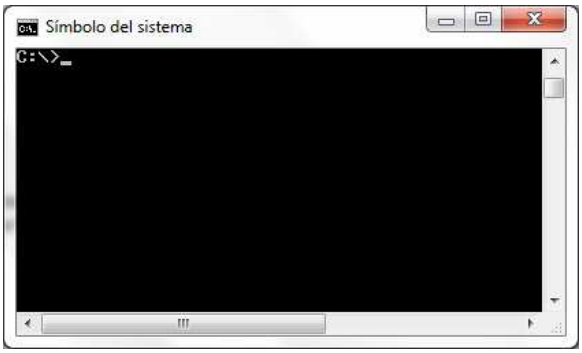


En la carpeta bin creamos un fichero llamado "Luz.manifest"
El contenido de ese fichero es el siguiente:

```
Manifest-Version: 1.0
Name: pulsador/Luz.class
Java-Bean: True
```

Es MUY IMPORTANTE que se pulse ENTER después de la última línea del fichero.

A continuación, abrimos una ventana de símbolo del sistema.



Nos situamos dentro de la carpeta bin y ejecutamos el siguiente comando:

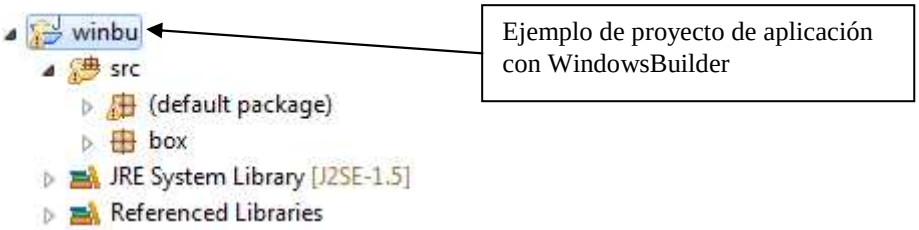
```
jar cfm Luz.jar Luz.manifest pulsador
```

Nos creará un fichero "Luz.jar" que contendrá nuestro componente y el manifiesto con la información del bean.
Este fichero podemos distribuirlo a otras personas que quieran utilizar nuestro componente.

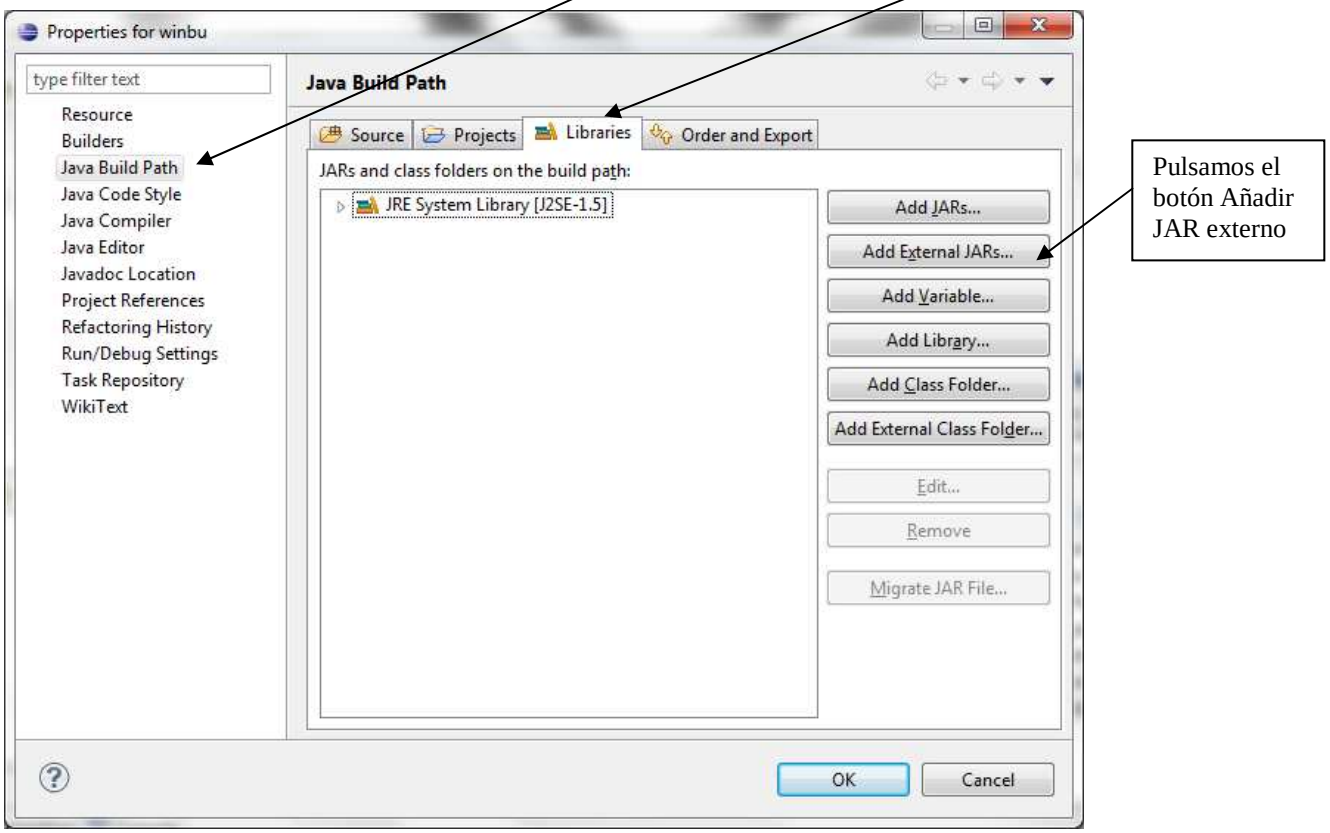
Añadir nuestro componente a uno de nuestros proyectos

Para poder utilizar el componente luz dentro de un proyecto, debemos añadir el fichero ".jar" que hemos creado al "Java Build Path" del proyecto. En el caso de que queramos utilizar el IDE de WindowsBuilder, éste, será un proyecto de WindowsBuilder.

En la ventana del Explorador de Paquetes de Eclipse pulsamos el botón derecho sobre la carpeta de nuestro proyecto.



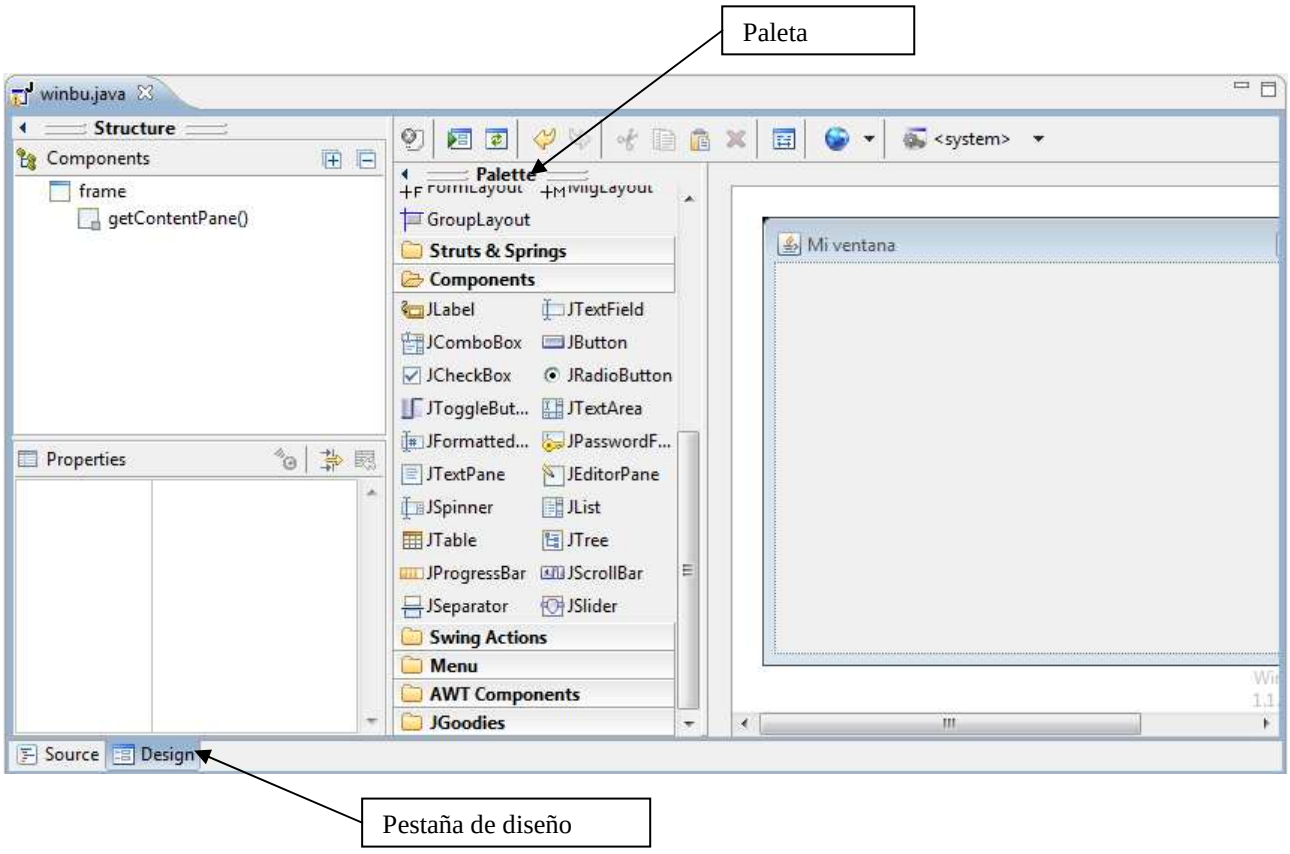
Al pulsar el botón derecho se abre un menú desplegable. Seleccionamos la opción "Properties..."
En las propiedades del proyecto seleccionamos la opción "Java Build Path" y la pestaña "Libraries"



Al pulsar el botón "Add External JARs" podemos navegar por las carpetas hasta nuestro fichero Luz.jar y seleccionarlo, con lo que quedará añadido a la biblioteca de clases del proyecto.

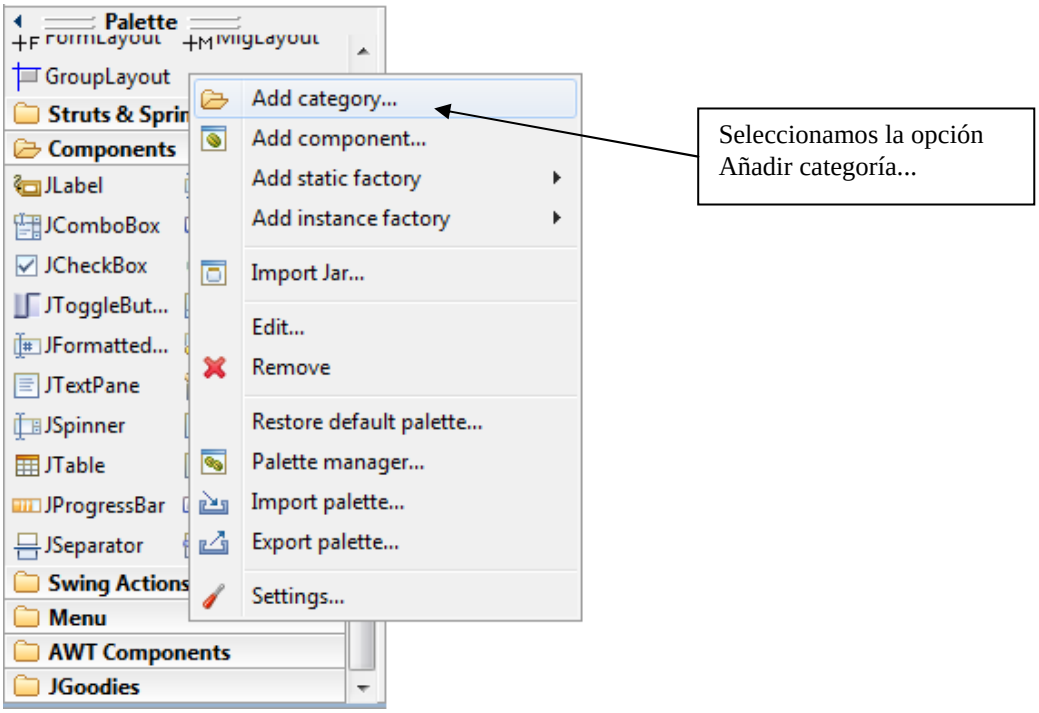
Colocando el componente en la paleta de un IDE (WindowsBuilder)

Abrimos nuestro proyecto de WindowsBuilder y seleccionamos la pestaña Design para acceder a la paleta del IDE

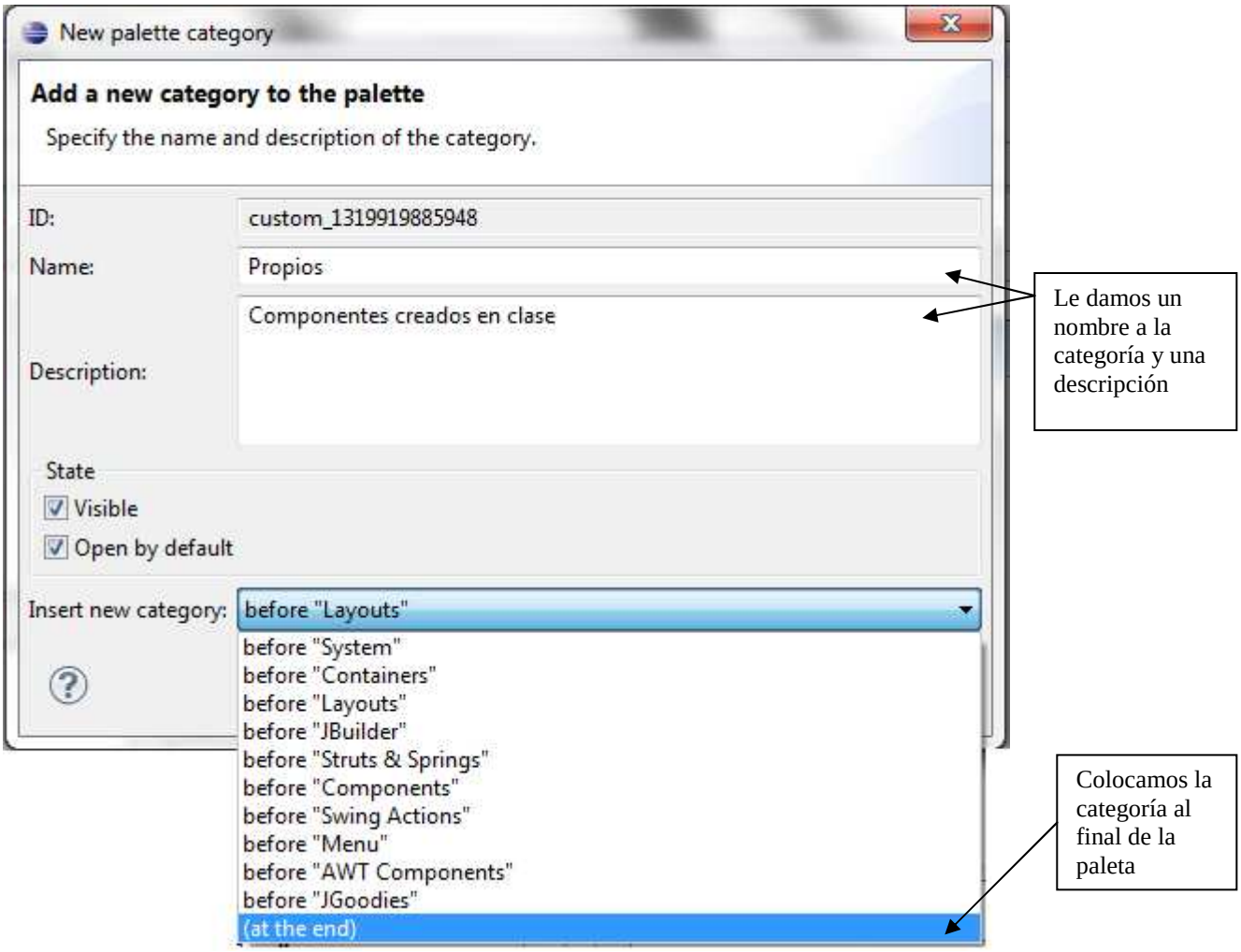


Creación de una categoría nueva

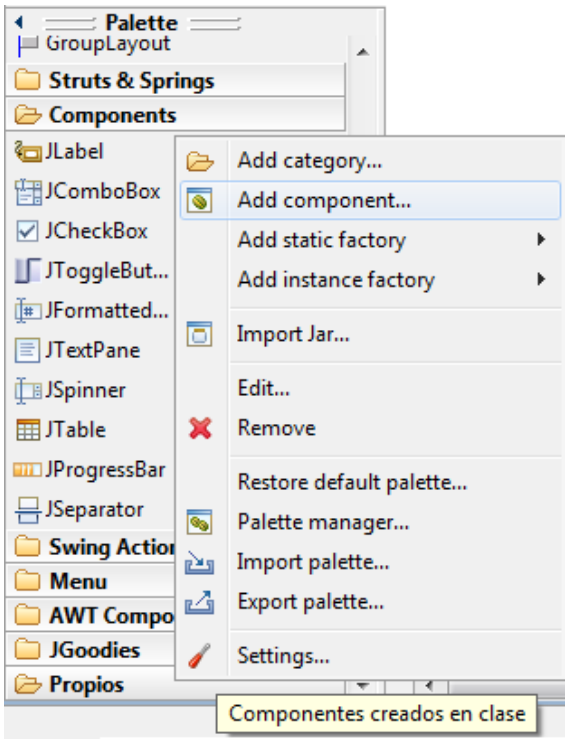
Pulsamos ahora el botón derecho sobre la Paleta y aparecerá un menú desplegable.



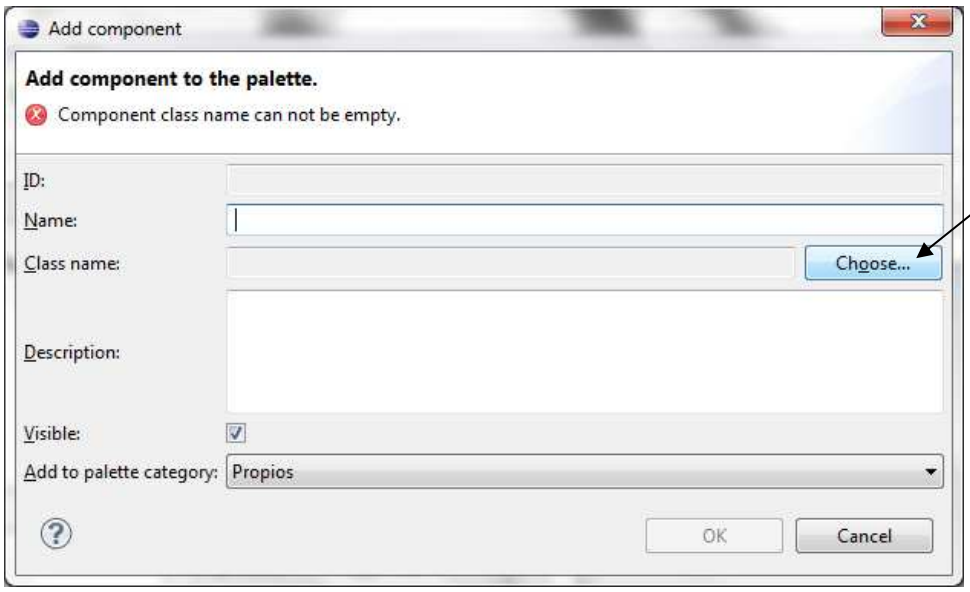
Vamos a crear una categoría de componentes nuevos en la paleta, donde situar nuestros componentes. Al seleccionar la opción "Add category..." se abrirá una ventana nueva.



Añadiendo el componente a la categoría

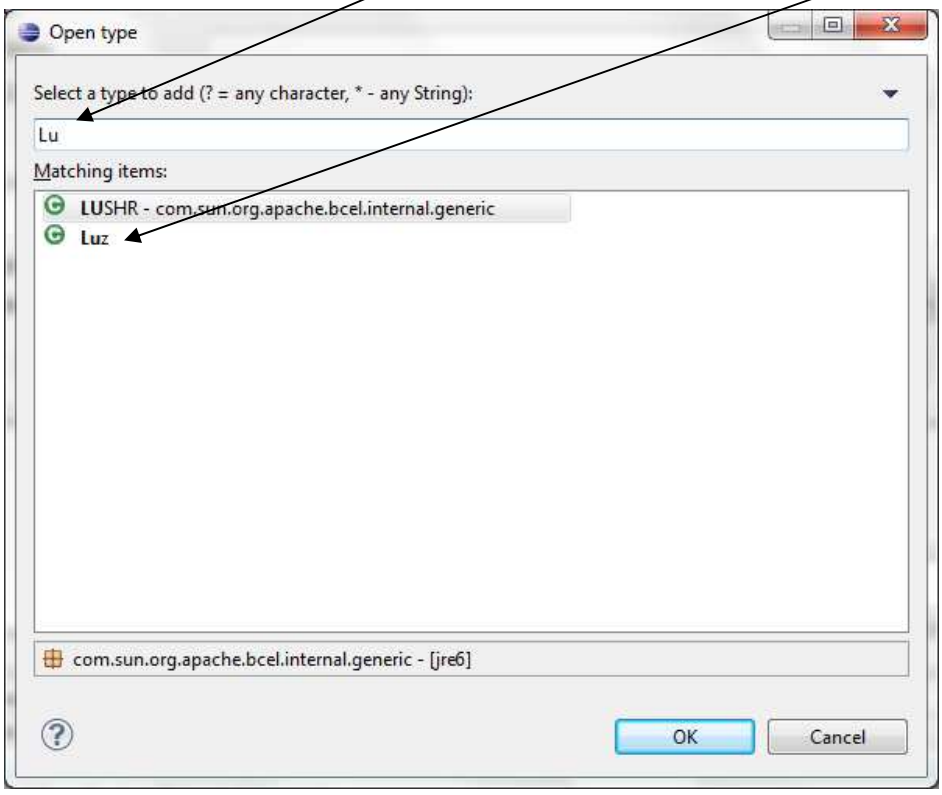


Pulsamos el botón derecho sobre la categoría "Propios" que hemos creado, y seleccionamos la opción "Add component..." del menú desplegable.



Pulsamos el botón Choose... para elegir el componente que queremos añadir a la paleta

Se abrirá una ventana para buscar el componente.
Escribimos el nombre del componente (Luz) para filtrar la búsqueda y lo seleccionamos y pulsamos el botón Ok.



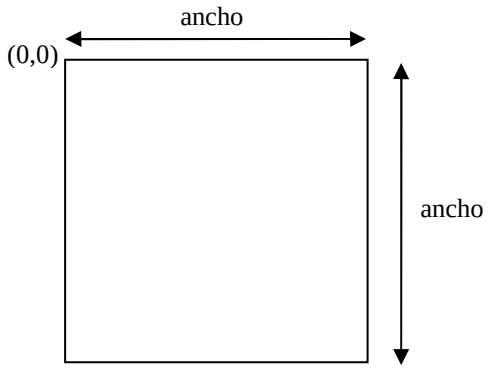
El componente se añade a la paleta con un icono genérico.

Probando el componente

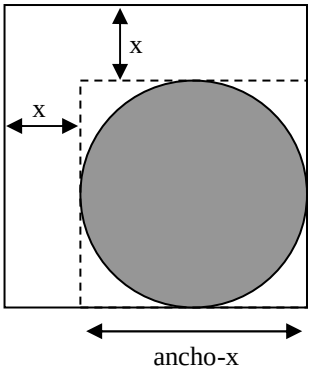
Se propone como ejercicio al alumno que diseñe un formulario sencillo con WindowsBuilder Pro usando un layout nulo (Absolute Layout), colocando varias instancias del componente pulsador, de distinto tamaño, cambiando también las propiedades "Nombre" y "Color".
En un elemento JTextField debería de aparecer un mensaje indicando cuándo cambia el estado de cada pulsador (y si queda activado o desactivado).

ANEXO A-Diseño gráfico del Bean

Los límites del Bean están determinados por un cuadrado de lado "ancho" que no se dibujará. La coordenada (0,0) corresponde con la esquina superior izquierda.

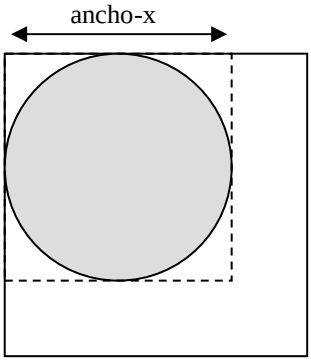


El dibujo del bean está formado por dos "tapas".
Primero dibujamos la tapa inferior del pulsador (un círculo con color gris oscuro)



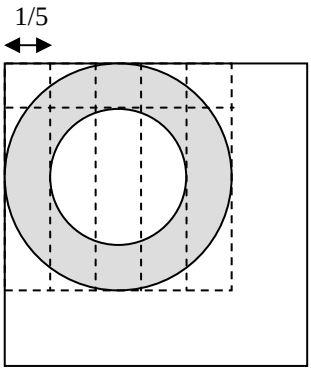
- con fillOval() dibujamos un círculo gris
- con drawOval lo perfilamos con una circunferencia negra

Luego dibujamos la tapa superior. La tapa superior es un círculo de color gris claro, de las mismas dimensiones que la tapa inferior, pero situado en la esquina superior izquierda.



- con fillOval() dibujamos un círculo gris claro
- con drawOval lo perfilamos con una circunferencia negra

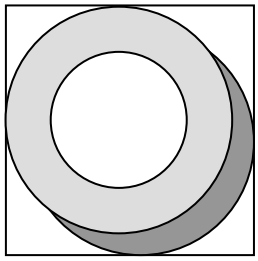
Finalmente dibujamos la luz del interruptor, como un círculo concéntrico



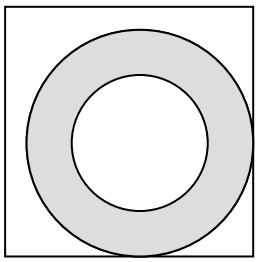
La diferencia de los radios de las circunferencias concéntricas es 1/5 de la anchura de la tapa

- con fillOval() dibujamos un círculo del color de la luz
- con drawOval lo perfilamos con una circunferencia negra

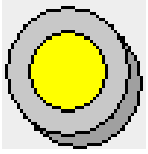
Si superponemos las "tapas" tenemos el dibujo del interruptor sin pulsar.



Para dibujar el interruptor pulsado, basta con hacer coincidir la tapa superior con la tapa inferior



El interruptor terminado



Valores en unidades. Para la práctica tomaremos un valor por defecto de 30 unidades para el "ancho" del botón, y un valor de 3 unidades para la "x" (coincidirá con el grosor del botón, es decir la distancia entre las tapas)

ancho=30
x=3