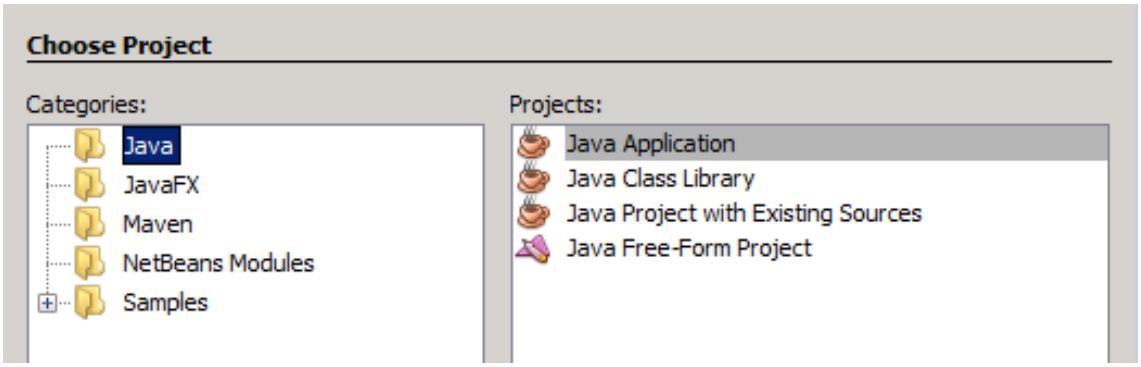
Práctica 0:Introducción a NetBeans

1 NetBeans.

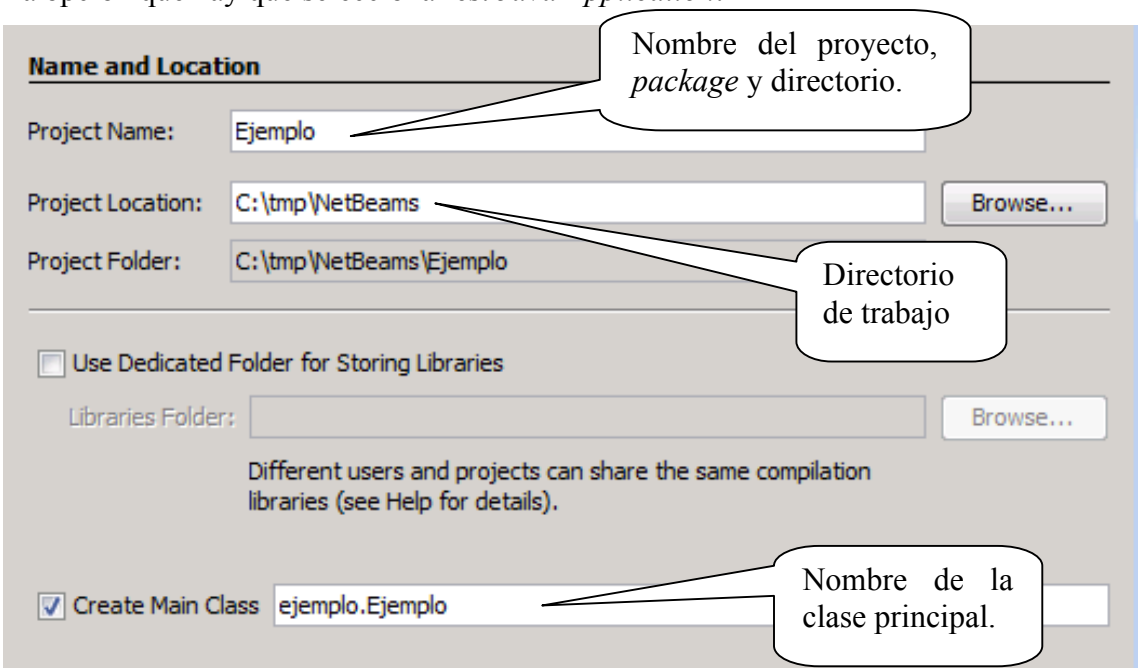
NetBeans (http://netbeans.org/) es un entorno de desarrollo multilenguaje. Permite, entre otras tareas, la creación interfaces gráficos de usuario basados en ventanas, que será lo que se use en esta práctica. Este documento tratará sobre la versión 7.2.

2 Creación de un proyecto.

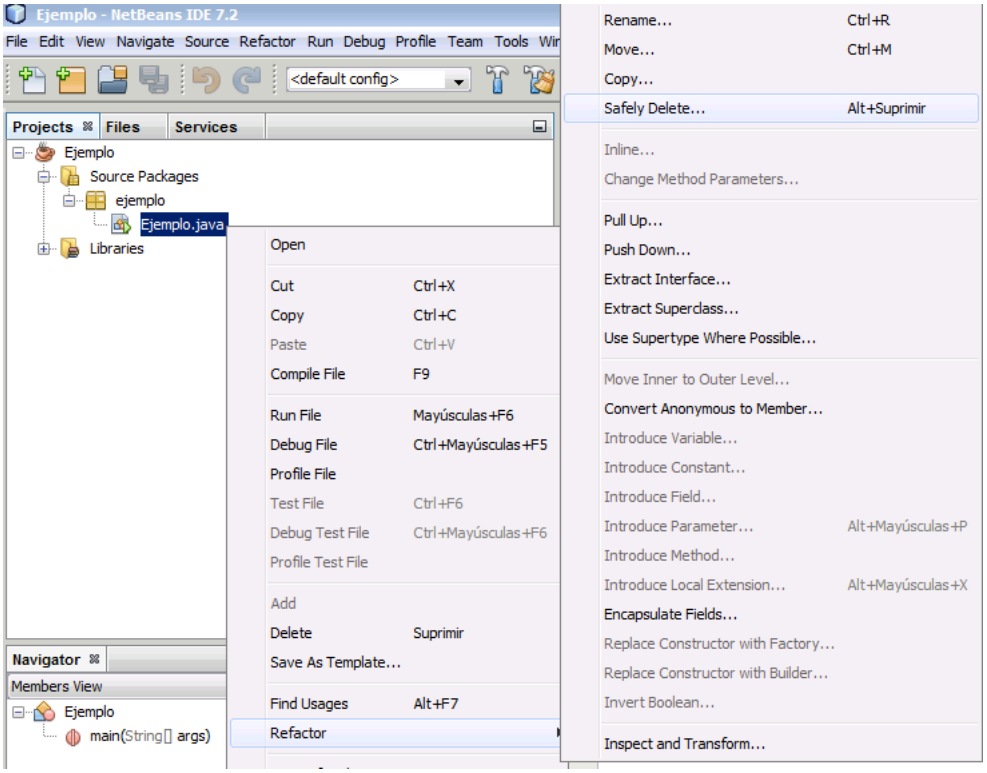
En el menú *File→New Proyect* ... aparece:



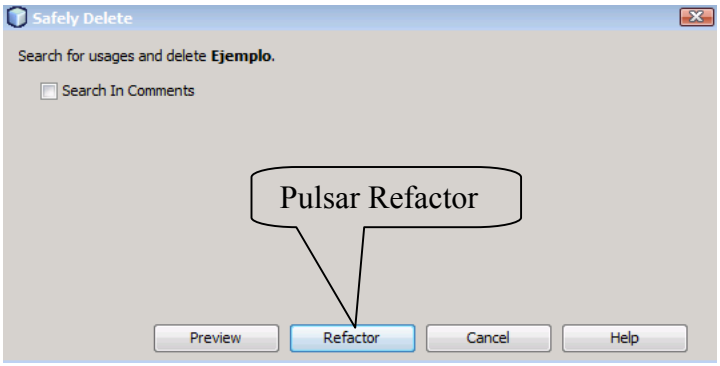
La opción que hay que seleccionar es: *Java Application*.



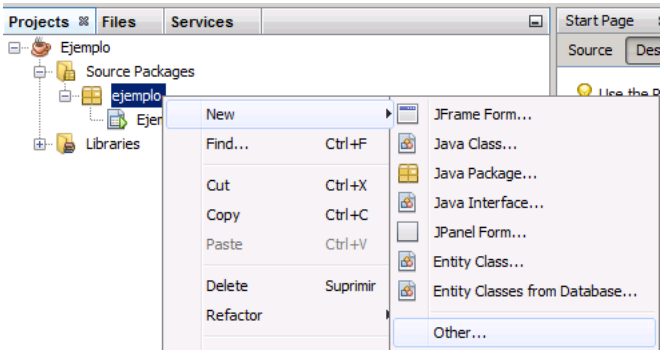
Se creará un proyecto con una clase principal que no acepta interface gráfico. Hay que eliminar esa clase y crear otra que admita dicho interface.



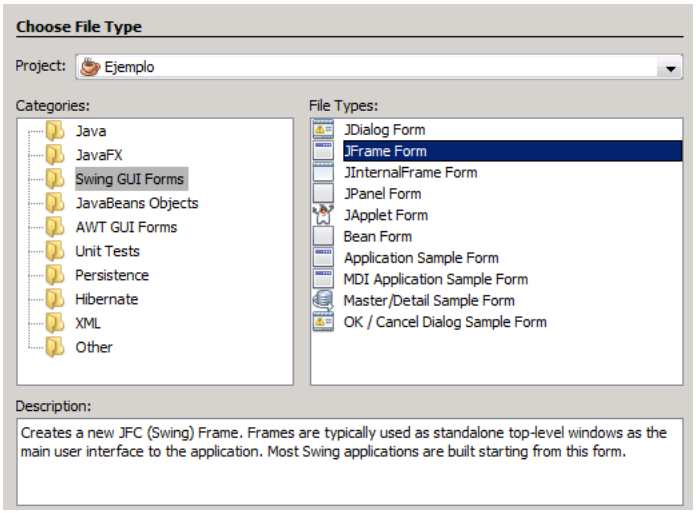
Con el menú de contexto (botón derecho del teclado en Windows) seleccionar *Refactor* y luego *Safely Delete*.



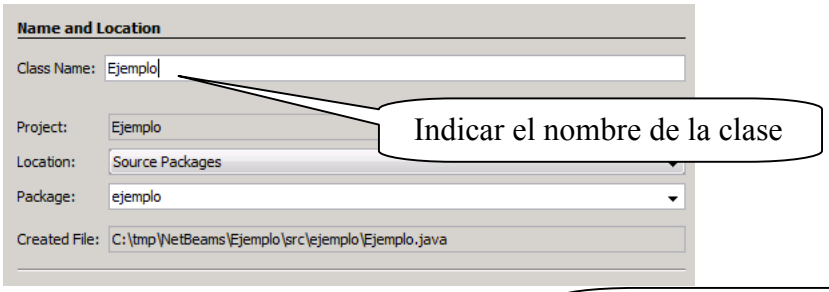
Se crea una nueva clase en el *package* que se ha creado.



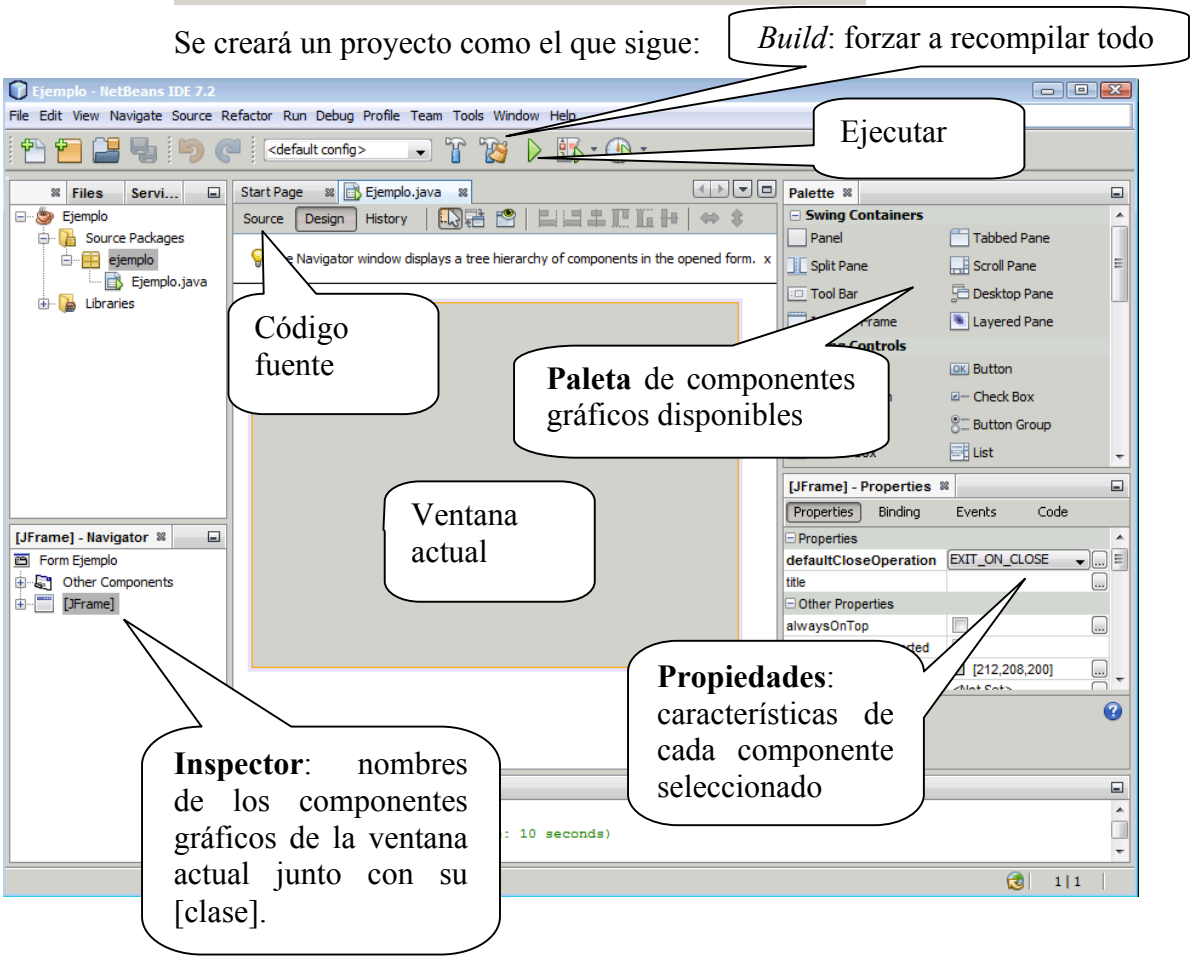
En el menú de contexto sobre el *package*, *new* → *Other*.



Dentro de *Swing GUI Forms*, elegir *JFrame Form*.



Se creará un proyecto como el que sigue:



Práctica 1: Ventana, componentes y sus eventos.

3 La ventana.

La ventana será un objeto de clase *JFrame*.

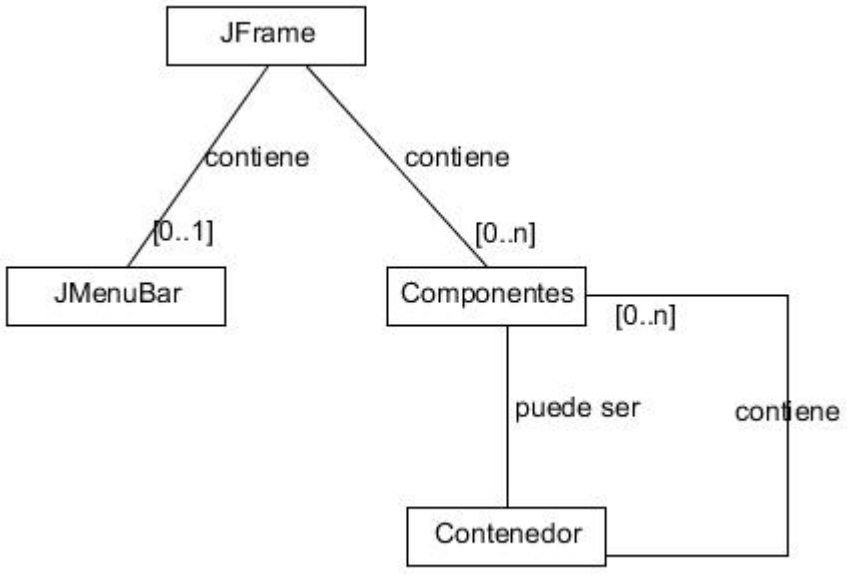
El tamaño y posición de la ventana en el IDE *Netbeans* será el que tenga inicialmente en ejecución.

La propiedad *title* indica el texto que aparecerá asociado a dicha ventana.

Pulsar *Ejecutar* (*Play*) para comprobar como queda la ventana en ejecución.

Sobre la ventana se pueden arrastrar y dejar otros componentes de la paleta. Seleccionado un componente se pueden modificar sus propiedades. La propiedad más importante es *Nombre* que define el nombre del componente. En el menú de contexto también se pueden modificar algunas propiedades y en algunos casos de manera más rápida.

Hay un componente especial que es la barra de menú (*JMenuBar*) solo puede haber 1 en cada ventana. Los componentes pueden ser de tipo Contenedor, éstos pueden contener otros componentes. El siguiente esquema sirve para ilustrar la organización de los componentes en la ventana:



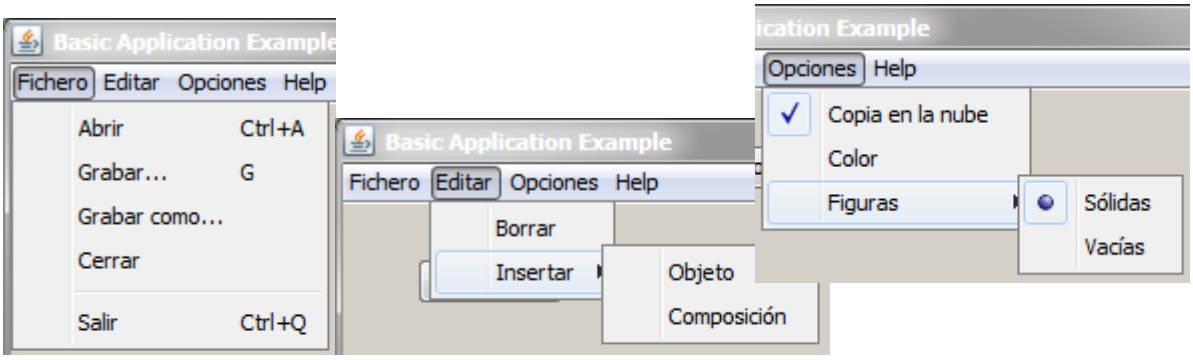
4 Los menús y sus propiedades.

En cada ventana se puede colocar una barra de menús (JMenuBar) y sobre ésta pueden colocar menús. Los menús pueden contener otros menús o ítems que son los elementos finales.

En el caso del menú se pueden utilizar las opciones del menú de contexto:

- Edit text
- Add From Palette
  - *Menu* → Nuevo menú o submenú
  - *Menu Item* → Nueva opción en un menú
  - *Menu Item* / *CheckBox* → Opción de activación independiente
  - *Menu Item* / *RadioButton* → Opción de selección excluyente
  - *Separator* → *Barra de separación*

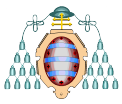
Actividad: crear una barra de menús como la que sigue:



Los (*Item) RadioButton* deben pertenecer a un conjunto para que cuando uno se active se sepa cuales hay que desactivar. Para agrupar varios RadioButton en un conjunto se pueden seguir los siguientes pasos:

1) Crear un *Button Group* (arrastrarlo a la ventana actual)
2) Cambiarle el nombre en el inspector (en la ventana principal no aparece), por ejemplo *BGMenuFiguras*.
3) Modificar la propiedad *buttonGroup* de cada *Item RadioButton* indicándole el grupo anterior.
4) Probar de nuevo y comprobar como al activar un *RadioButton* se desactiva el otro.

Actividad: Añadir otro menú: `Colores`→`{Pastel; Vivos}` mediante *RadioButtons* pero independientes de los anteriores (utilizar otro *buttonGroup*).



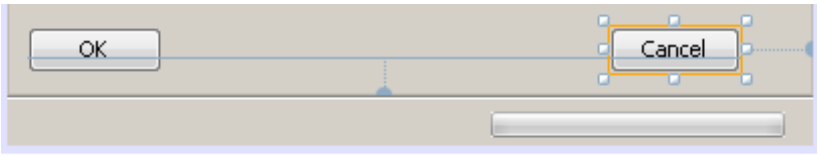
5 Componentes en la ventana y su disposición (layout)

La barra de menús es independiente del *layout*, pero colocar otros componentes en la ventana hay que arrastrarlos a la misma, pero su disposición final dependerá del *layout* sobre el que se arrastre.

5.1 *Panel* layout.

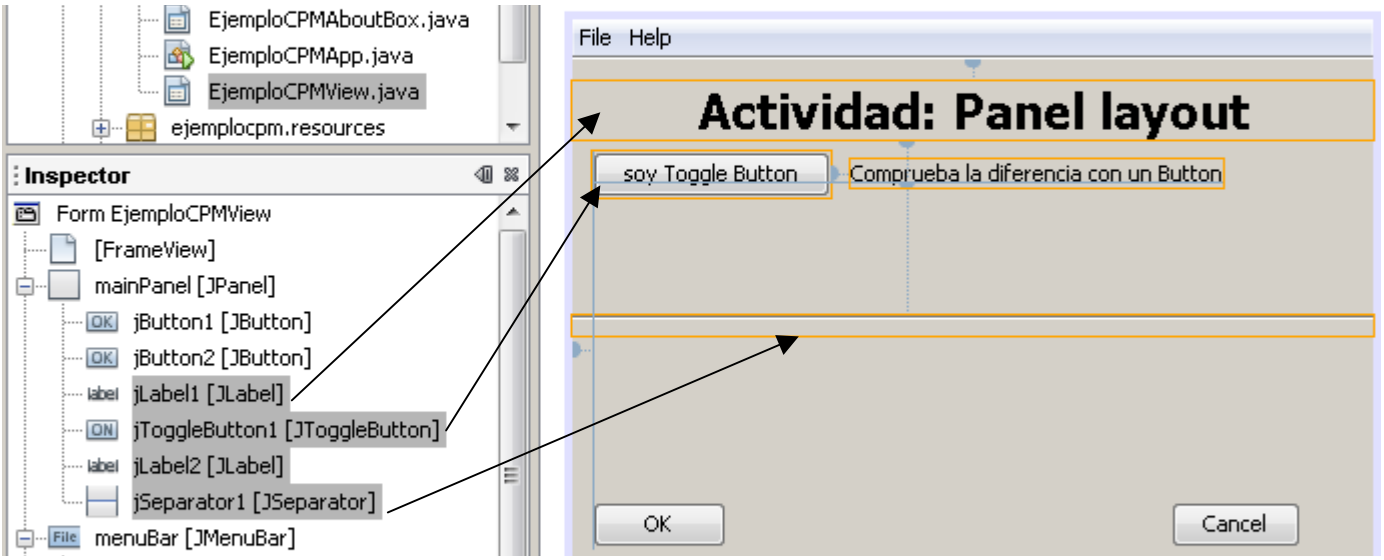
El panel por defecto es de clase *Panel*. El layout permite situar objetos en posiciones relativas a otros. El interfaz lo muestra con unas líneas continuas de posición y unas líneas discontinuas que marcan las distancias que se han de cumplir.

Actividad: colocar dos botones: OK y Cancel en la parte inferior y alineados horizontalmente.



Ejecutar y cambiar el tamaño de la ventana. ¿Cómo se mueven los botones respecto a la ventana? ¿Es coherente con las líneas de posición y distancias?

Actividad: crear una interfaz similar a esta:

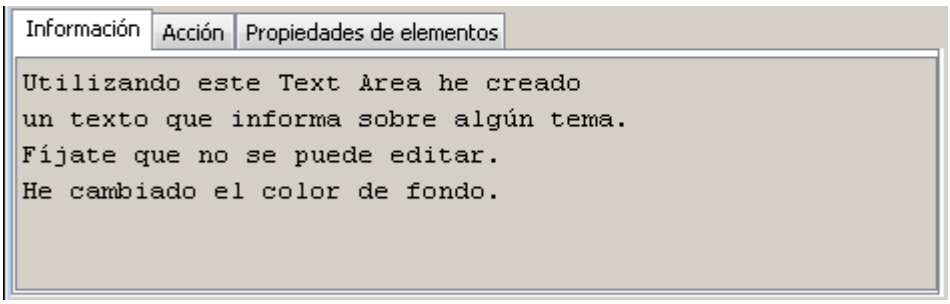


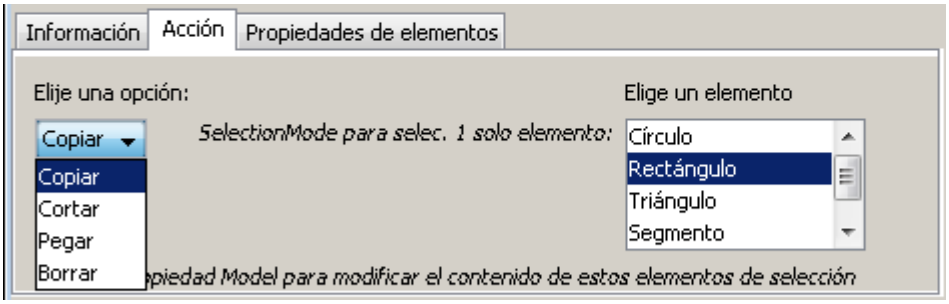
1. Haz que la etiqueta (*label*) 1 tenga el texto (*font*) grande (Tahoma 24 bold) y esté centrado (*HorizontalAlignment*).
2. Haz que la label 1 y el separador vayan de extremo a extremo de la ventana.
3. Ejecuta y comprueba que pasa cuando se modifica el tamaño de la ventana.
4. Comprueba que ocurre si en el separador se desactiva *HorizontalResizable*.

5.2 *Tabbed* y *Split* layouts.

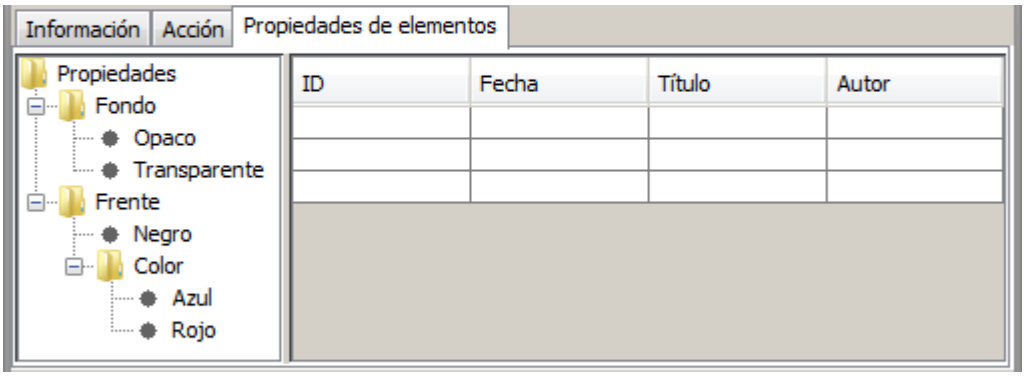
Es un meta-layout porque permite incluir dentro de éste varios layouts, así como componentes individuales. Para incluir un layout o elemento hay que arrastrarlo sobre la parte superior del *Tabbed Pane* el panel que se desea incluir, aparecerá una línea discontinua naranja. La opción *Edit Text* del *Tabbed Pane* permite cambiar la etiqueta de la pestaña activa.

Actividad: crear una configuración de ventanas en pestañas como la que sigue:





Utilizar un *Split Pane* para la siguiente pestaña:



Un *Split Pane* funciona igual que un *Tabbed Pane* pero solo tiene dos elementos y se muestran a la vez.

Aunque por defecto tenga dos botones se puede arrastrar cualquier otro elemento.

5.3 *Scroll* y *Layered* layouts.

El *Scroll Pane* permite disponer de elementos que ocupen más que el espacio de que se dispone. En este caso aparecerán barras de desplazamiento que permiten moverse por el elemento.

Es necesario arrastrar al *Scroll Pane* un elemento, que suele ser otro panel.

El *Layered Pane* permite situar elementos de manera absoluta sobre la superficie del mismo, a diferencia del *Panel* que los sitúa de manera relativa.

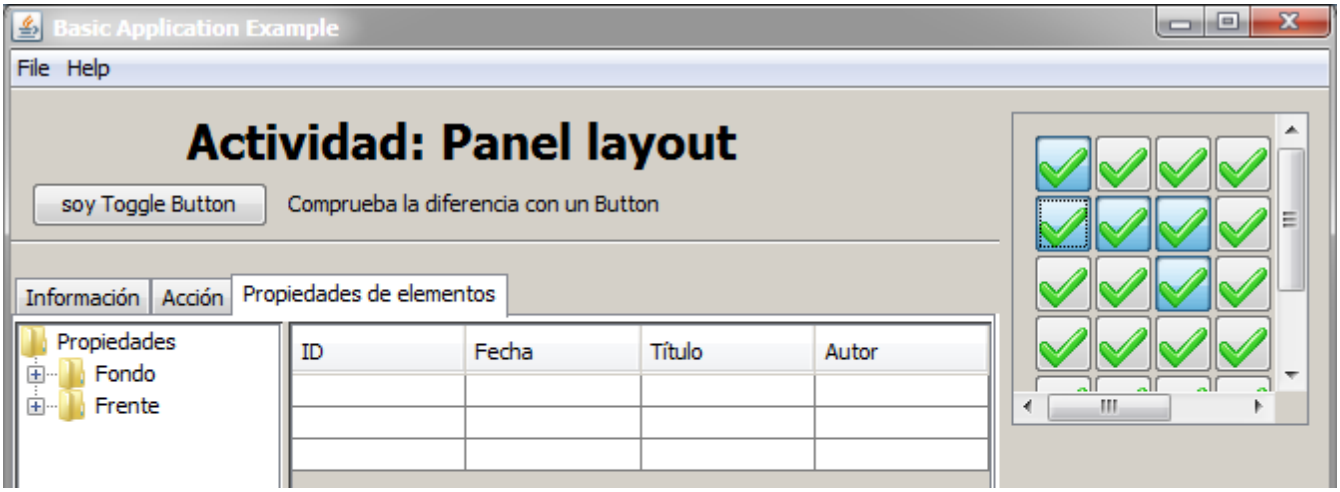
Actividad: crear un *Layered* layout (grande) dentro de un *Scroll* layout (pequeño) de manera que tenga la siguiente apariencia:

Crear un botón con imagen:

1. Obtener la imagen:
    a. Buscar “icons png” en Google images.
    b. http://www.small-icons.com/packs/24x24-free-application-icons.htm
2. Copiar la imagen seleccionada en: `src/<package>/resource`
3. Crear un botón y modificar propiedad “icon”

Se indica el tamaño del *Layered Pane* utilizando la propiedad “preferedSize”.

Se puede hacer doble clic en un elemento para editarlo solo (puede ser sobre la imagen del elemento o sobre su nombre en el inspector).

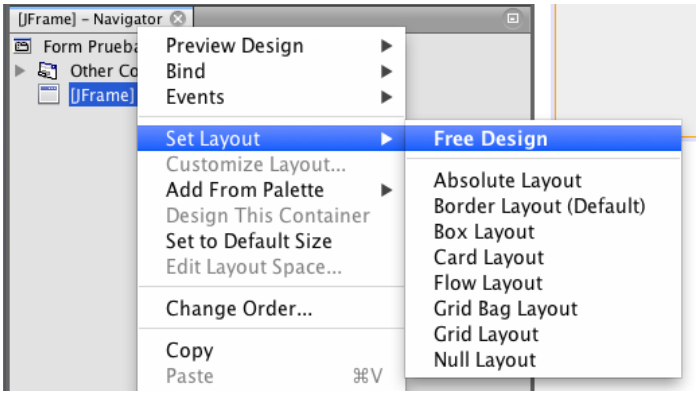


Crear 6x6 botones (mira la imagen ¿qué tipo de botones?) de 30x30 cada 30 unidades y empezando en 10 (Mirar el apartado layout de las propiedades del botón). Crear uno y luego copiar y pegar el resto.

NOTA: en la versión 7.0 cuando copias un botón no se copia la propiedad "icon", pero el interfaz de desarrollo sí la copia. En ejecución desaparece. Se pueden seleccionar todos y luego dar valor a la propiedad "icon".

6 Cambiar el layout de una ventana.

Obtener el menú de contexto sobre `[jFrame]`, menú `Set Layout` y elegir un *layout*.



Probar con diferentes tipos de *layout*.

Para utilizar un *swing container* como *layout*, elegir *Grid Layout* y arrastrar el *container* desado.

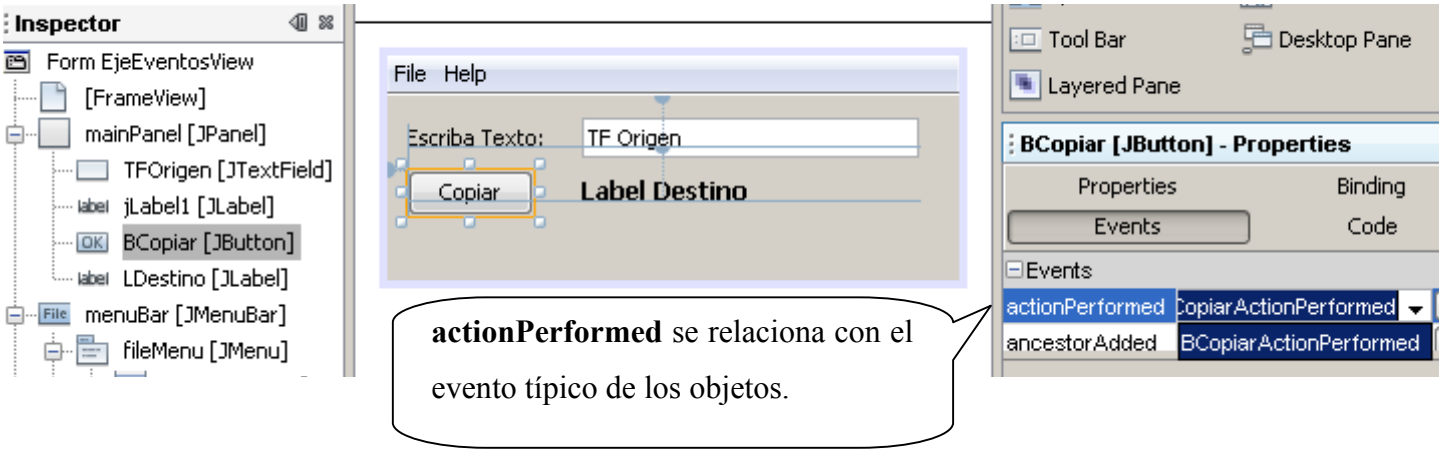
7 Cambiar el título de la aplicación.

En la ventana de código fuente, se selecciona el proyecto y mediante el menú de contexto se modifican las propiedades. En *Application* → *Title*, se puede indicar el título de la aplicación.

8 Eventos.

Se puede asignar una acción un evento seleccionando el objeto y luego seleccionando *Events*. En ese momento aparecerá la lista de eventos. Eligiendo el evento al que se desea indicar la acción se elige en el *ComboBox* asociado el método donde se codificará la acción (aparece un nombre por defecto) y pasamos al modo *Source*. En este modo se muestra el código java en vez del diseño de la interfaz.

Actividad: crear un interfaz con el diseño y variables como se indica a continuación:



9 Acciones.

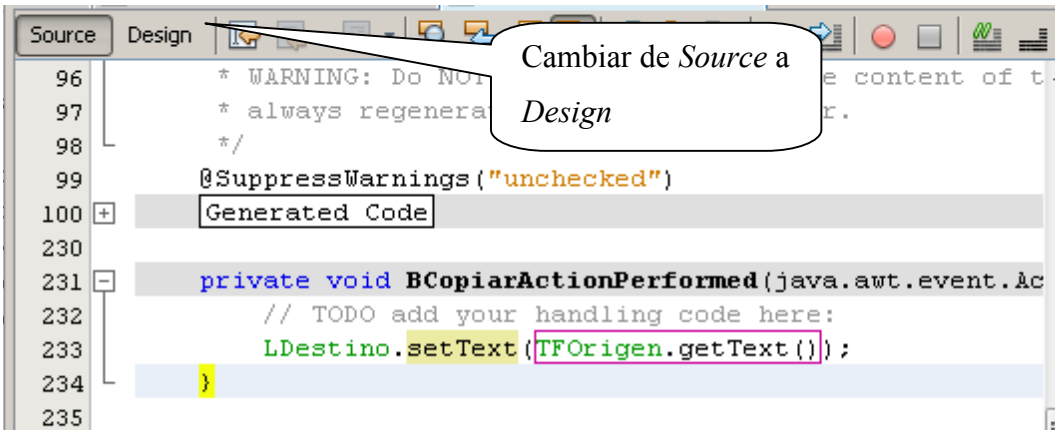
En el modo *Source* se escribe el código que implementa las acciones. En general para obtener un valor de una propiedad de un objeto utilizaremos métodos *get* y para modificar el valor utilizaremos métodos *set*.

```
<NombreDeObjeto>.get<NombreDePropiedad>();
<NombreDeObjeto>.set<NombreDePorpiedad>(<Valor>);
```

En algunos casos más complicados será necesario utilizar otros métodos, los cuales siempre los encontraremos descritos en la ayuda que proporciona java sobre estas clases: http://docs.oracle.com/javase/8/docs/api/.

NetBeans ofrece la posibilidad de autocompletar nombres de métodos o campos, lo que facilita encontrar los métodos y saber sus parámetros y el tipo que retorna.

Ejemplo: Modificar la interfaz anterior para que al pulsar el botón se copie el texto del *TextForm* origen a la *Label* destino.



10 Eventos de teclado.

En componentes que admitan texto, los principales eventos son:

- **KeyTyped**: se ejecuta después de teclear un carácter y antes de que el evento modifique el objeto.
- **KeyReleased**: se ejecuta después de que el objeto se haya modificado por introducir un carácter.

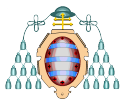
Actividad: modificar la interfaz anterior para que cada vez que se modifique el origen cambie el destino. ¿Cuál de los tres eventos hay que utilizar?

10.1 Filtros de teclado.

El evento *KeyTyped* permite modificar o eliminar el carácter tecleado. Para esto se modificará el parámetro *evt* de clase *KeyEvent* en el método que recibe el evento.

Los métodos de la clase KeyEvent para obtener el carácter y modificarlo son respectivamente: `char getKeyChar()` y `setKeyChar(char c)`. Para eliminar el carácter se puede utilizar: `evt.consume();`.

Actividad: modificar el interfaz anterior para que el origen convierta cualquier carácter en mayúsculas. Mirar en la ayuda cómo utilizar la clase *Character* para pasar un carácter a mayúsculas.



11 Listas.

11.1 Acceso a la selección en las listas.

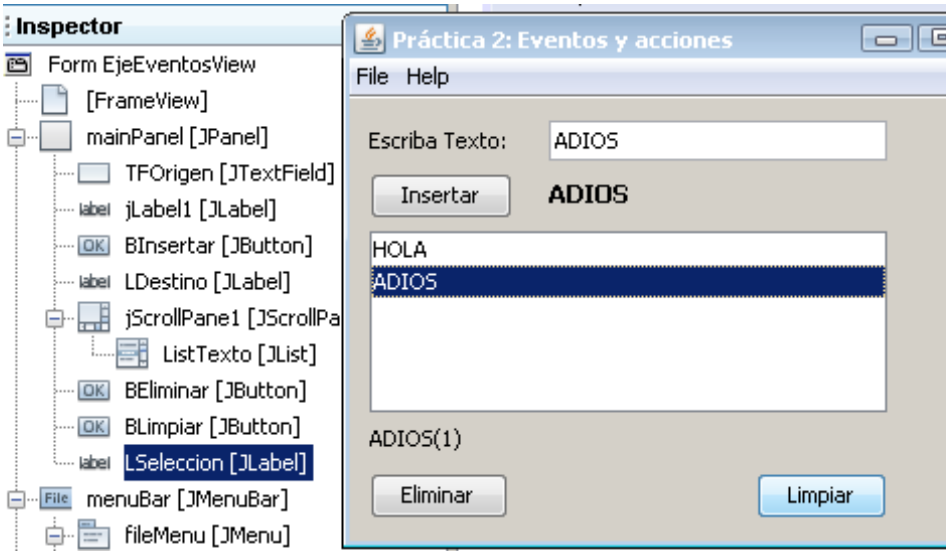
Cuando un lista solo puede seleccionar un elemento, los métodos:

```
Object getSelectedValue()
int getSelectedIndex()
```

Permiten obtener el objeto seleccionado y su posición en la lista.

El evento *ValueChanged* se genera cuando ha cambiado la selección.

Actividad: Crear un interface y las variables como el que sigue, en el que se modifique una label al seleccionar un elemento de la lista indicando en la label la cadena asociada al elemento seleccionado y su posición en la lista.



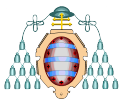
11.2 Inserción en listas.

Las listas que se crean en el NetBean IDE están pensadas para ser estáticas, si se quiere que se modifiquen dinámicamente hay que modificar la clase de la propiedad *Model* con algún objeto que cumpla `javax.swing.ListModel`.

Para inicializar cualquier objeto creado con el IDE se recomienda que se haga en el constructor de la ventana, justo después de la llamada `initComponents();`, ejemplo:

```
initComponents();

// Crear un Model del ListBox que no sea estático
ListTexto.setModel(new javax.swing.DefaultListModel());
```



Una vez que se tenga creado un modelo del tipo deseado, para modificarlo, se utilizará el método *getModel* y con un casting lo convertiremos al tipo del que fue creado.

En el caso de un modelo de la clase `DefaultListModel` se podrá utilizar el método `addElement` para insertar un nuevo elemento en el modelo.

Actividad: modificar el interfaz anterior permitiendo que el botón *Insertar*, inserte el contenido del `TextField` en la lista.

11.3 Borrado de listas.

Para borrar un elemento en una lista hay que seguir tres pasos:

1. Obtener el índice del elemento que se quiere borrar.
2. Obtener el `DefaultListModel` que contiene la lista de elementos.
3. Invocar el método `void remove(int index)` del `DefaultListModel`.

NOTA: si se borra el elemento seleccionado, no se selecciona el siguiente.

Si lo que se quiere es borrar es el elemento seleccionado hay que comprobar que existe selección (utilizando el método de la *JList* `boolean isSelectionEmpty()`) y luego obtener el índice como se mostró un apartado anterior.

Si se quiere borrar toda la lista se puede utilizar el método `void clear()` de la clase `DefaultListModel`.

Actividad: modificar la interfaz anterior para que funcionen los botones *Eliminar* (elimina el elemento de la lista seleccionado) y *Limpiar* (borra todos los elementos de la lista).

11.3.1 Mantener selección al borrar en listas.

Si se quiere modificar la selección para que al borrar el elemento seleccionado, el siguiente de la lista pase a ser el seleccionado habría que utilizar el método de la clase *JList* `void setSelectedIndex(int index)`. Tras el borrado hay que calcular cual es el nuevo índice de la lista que tiene que estar seleccionado. Para saber si se ha borrado el último elemento de la lista se puede utilizar el método `void size()` del `DefaultListModel`.

Actividad: modificar la interfaz anterior para que el borrado mantenga la selección.