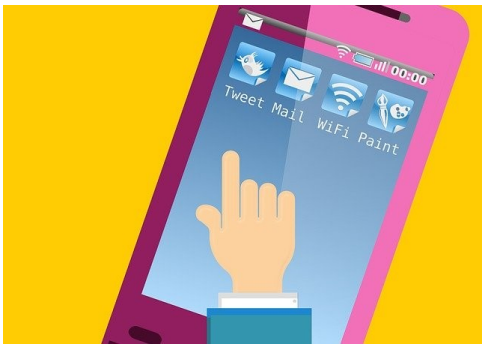
Confección de interfaces de usuario

ÍNDICE

1.- Elaboración de interfaces de usuario.

El desarrollo de aplicaciones hoy día no puede ser entendido sin dedicar un porcentaje bastante importante de tiempo a planificar, analizar, diseñar, implementar y probar sus interfaces de usuario ya que son el medio fundamental por el cual el usuario puede comunicarse con la aplicación y realizar las operaciones para las que ha sido diseñada.



Gerd Altmann (Dominio público)

Existen diferentes tipos de interfaces de usuario:

- Textuales. La comunicación se produce por medio de la inserción de órdenes escritas en un intérprete de órdenes.
- Gráficas. La interfaz consta de un conjunto de elementos visuales, como iconos o menús con los que se interacciona, normalmente, mediante un elemento apuntador (el ratón, por ejemplo). Son las más habituales y tienen a gala haber popularizado el mundo de la informática para usuarios noveles.
- Táctiles. La comunicación se produce mediante a través de un dispositivo táctil, generalmente una pantalla que puede reaccionar ante la presión táctil de elementos apuntadores o incluso de los dedos. Se usan habitualmente en dispositivos móviles, terminales de puntos de venta y para el diseño de gráficos por ordenador.

A lo largo de esta unidad nos centraremos en la creación de interfaces gráficas. Veremos que una interfaz gráfica está formada por un conjunto de ventanas, llamadas formularios, y que dentro de ellos podemos colocar diferentes elementos visuales, que se denominan controles o componentes con los que al interactuar damos órdenes o podemos recibir información.

2.- Componentes.

Una interfaz gráfica se comporta como un todo para proporcionar un servicio al usuario permitiendo que éste realice peticiones, y mostrando el resultado de las acciones realizadas por la aplicación. Sin embargo, se compone de una serie de elementos gráficos atómicos que tiene sus propias características y funciones y que se combinan para formar la interfaz. A estos elementos se les llama **componentes** o **controles**.

Algunos de los componentes más típicos son:

- **Etiquetas**: Permiten situar un texto en la interfaz. No son interactivos y puede utilizarse para escribir texto en varias líneas.
- **Campos de texto:** cuadros de una sola línea en los que podemos escribir algún dato.

- **Áreas de texto:** cuadros de varias líneas en los que podemos escribir párrafos.
- **Botones:** áreas rectangulares que se pueden pulsar para llevar a cabo alguna acción.
- **Botones de radio:** botones circulares que se presentan agrupados para realizar una selección de un único elemento entre ellos. Se usan para preguntar un dato dentro de un conjunto. El botón marcado se representa mediante un círculo.
- **Cuadros de verificación:** botones en forma de rectángulo. Se usan para marcar una opción. Cuando está marcada aparece con un tic dentro de ella. Se pueden seleccionar varios en un conjunto.
- **Imágenes:** se usan para añadir información gráfica a la interfaz.
- **Password:** es un cuadro de texto en el que los caracteres aparecen ocultos. Se usa para escribir contraseñas que no deben ser vistas por otros usuarios.
- **Listas:** conjunto de datos que se presentan en un cuadro entre los que es posible elegir uno o varios.
- **Listas desplegables:** combinación de cuadro de texto y lista, permites escribir un dato o seleccionarlo de la lista que aparece oculta y puede ser desplegada.

2.1.- Bibliotecas de componentes.

Los componentes que pueden formar parte de una interfaz gráfica se suelen presentar agrupados en bibliotecas con la posibilidad de que el usuario pueda generar sus propios componentes y añadirlos o crear sus propias bibliotecas. Se componen de un conjunto de clases que se pueden incluir en proyectos software para crear interfaces gráficas. El uso de unas bibliotecas u otras depende de varios factores, uno de los más importantes, por supuesto, es el lenguaje de programación o el entorno de desarrollo que se vaya a usar. Dependiendo de esto podemos destacar:

JAVA Fundation Classes (JFC):

Las .E.9. incluyen las bibliotecas para crear las interfaces gráficas de las aplicaciones Java y applets de Java.

- **AWT:** Primera biblioteca de Java para la creación de interfaces gráficas. Es común a todas las plataformas, pero cada una tiene sus propios componentes, escritos en código nativo para ellos. Prácticamente en desuso.
- **Swing**: Surgida con posterioridad, sus componentes son totalmente multiplataforma porque no tienen nada de código nativo, tienen su precursor en A.fil, de hecho, muchos componentes swing derivan de AWT, basta con añadir una J al principio del nombre AWT para tener el nombre swing, por ejemplo, el elemento Button de AWT tiene su correspondencia swing en Jbutton aunque se han añadido gran cantidad de

componentes nuevos. Es el estándar actual para el desarrollo de interfaces gráficas en Java.

Además, existen bibliotecas para desarrollo gratifico en 2D y 3D y para realizar tareas de arrastrar y soltar (drag and drop).

Bibliotecas .NET de Microsoft (C#, ASP, ...):

- .NET framework: hace alusión tanto al componente integral que permite la compilación y ejecución de aplicaciones y webs como a la propia biblioteca de componentes que permite su creación. Para el desarrollo de interfaces gráficas la biblioteca incluye ADO.NET, ASP.NET, formularios Windows Forms y la WPF (Windows Presentation Fundation).

Bibliotecas basadas en XML:

- También existen bibliotecas implementadas en lenguajes intermedios basados en tecnologías XML. Normalmente disponen de mecanismos para elaborar las interfaces y traducirlas a diferentes lenguajes de programación, para después ser integradas en la aplicación final.

Otras API (Application Programming Interface, Interfaz de programación)

También hay que destacar que existen otras bibliotecas o API como son:

- **Directx**: plataforma Microsoft creada para facilitar el manejo de los elementos multimedia. Consta a su vez de otras API como son Direct3D, Direct Graphics, Direct sound, Direct Input, DirectPlay, DirectShow, DirectMusic, DirectSetuo y DirectCompute.
- **GTK (GIMP tOOL KIT}:** biblioteca del equipo GTK+. El entorno gráfico de GNOME utiliza esta librería. Maneja widgets como ventanas, etiquetes, pestañas, etc y se puede utilizar en lenguajes C, C++, C#, Java, Python, Ruby.
- **QT:** es utiliza por el entorno gráfico KDE. Utiliza lenguaje de programación C++ y puede ser integrado en otros lenguajes. Se utiliza porque también se utilizan en sistemas empotrados como automoción, aereonavegación y aparatos domésticos.

3.- Herramientas para la elaboración de interfaces.

Con las bibliotecas tienes la base para crear tus aplicaciones, están compuestas de código que puedes escribir sin más, aunque lo habitual es valerse de algún entorno integrado de desarrollo de software que facilite esta labor mediante algún sistema de ventanas, en el que se incluyen diferentes regiones rectangulares, llamadas "ventanas" cuya parte central es un conjunto de herramientas (toolkit). A este tipo de herramientas se le llama IDE (Integrated Development Environment) y provee de procedimientos que permiten incluir los

componentes de las bibliotecas de componentes gráficos y añadirlos de un modo sencillo e intuitivo en la aplicación que estés desarrollando. También sirven como medio de entrada de las acciones del usuario.

A continuación, se listan varios de los IDE más utilizados en la actualidad:

- **Microsoft Visual Studio:** Es un entorno para el desarrollo de aplicaciones en entornos de escritorio, web y dispositivos móviles con la biblioteca .NET framework de Microsoft. Permite el uso de diferentes lenguajes de programación como C++, C#,o ASP. En la actualidad se pueden crear aplicaciones para Windows 7, 8 y 1O, aplicaciones web y RIA (Rich Internet Aplications) .

- **NetBeans:** IDE distribuida por Oracle bajo licencia GNU GPL. Está desarrollado en Java, aunque permite crear aplicaCio.nes en diferentes lenguajes, Java, C++, PHP, Ajax, Python y otras.

- **Eclipse:** IDE creado inicialmente por IBM ahora es desarrollado por la fundación Eclipse. Se distribuye bajo la Licencia Pública Eclipse (EPL). Tiene como característica más destacable su ligereza ya que se basa en módulos, partiendo de una base muy sencilla con un editor de texto con resaltado de sintaxis, compilador, un módulo de pruebas unitarias con JUnit, control de versiones con CVS, integración con Ant, asistentes (wizards) para creación de proyectos, clases, tests, etc. y refactorización. Si se precisan funcionalidades más allá, éstas se incluirán como módulos aparte que vi n completando el IDE.

- **JDeveloper:** Es un entorno de desarrollo integrado desarrollado por Orade··corporation para los lenguajes Java, HTML, XML, SQL, PL/SQL, Javascript, PHP, Oracle ADF, UML y otros.

- **Aptana Studio:** basado en eclipse incluye un importante motor para el desarrollo de interfaces web.

- **Oracle Database:** permite crear interfaces para acceder a bases de datos.

- **Dreamweaver:** no solo permite el diseño de interfaces web sino que también permite el desarrollo de aplicaciones basadas en ASP.NET, PHP, Javascript, CSS, ColdFusion, etc.

- **Komodo Edit:** sus características dependen de un intérprete de phytori

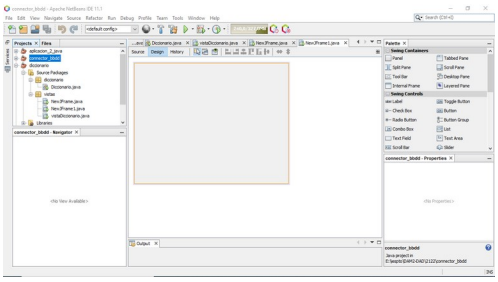
3.1.- NetBeans.

Para desarrollar los contenidos de esta unidad, te proponemos NetBeans como entorno de desarrollo integrado, utilizando la biblioteca swing para la generación de interfaces. Se ha seleccionado este entorno por varios motivos, en primer lugar, permite crear aplicaciones tanto de escritorio, como web como para dispositivos móviles y además se distribuye bajo licencia CDDL y GPL2. Es multiplataforma e incluye varios lenguajes de programación.

Una de sus principales ventajas es que resuelve por sí mismo el tema de la colocación de los componentes en una interfaz gráfica, aspecto de cierta complejidad a la hora de programar, de forma que el desarrollador o desarrolladora sólo tiene que colocar los controles usando el ratón y el IDE. se encarga de programarlo.

También permite la inclusión de componentes creados por otros desarrolladores que completan la paleta swing/AWT.

Ejemplo proyecto "AplicacionEscritorio"

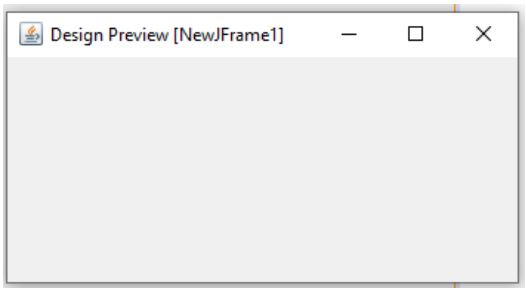


Una vez que tengas el entorno instalado y funcionando, debes comenzar por crear un proyecto que pueda utilizar las clases incluidas en la librería swing, para ello sólo debes seleccionar como tipo de proyecto **Java** y en tipo de proyecto seleccionar Java Application. A continuación, rellena los datos principales del proyecto, como su nombre, la localización de los archivos en disco y si queremos crear una clase principal (la activaremos). A continuación, pulsamos terminar y tenemos creado un proyecto nuevo. A continuación, pulsaremos el boto derecho del ratón (a esto se le llama menú contextual) sobre la opción paquetes de fuentes (Source packages) y elegiremos Nuevo / Formularios de interfaz gráfica swing (Swing GUI Forms) / Formulario JFrame (Jframe Forms) / escribiremos el nombre del nuevo JFrame, en nuestro caso, Formulario y pulsaremos terminar.

Para poder ejecutar el proyecto y visualizar el formulario, debes de indicar cual va a ser la clase principal. Para ello, seleccionamos el proyecto en la ventana Proyectos, pulsas el botón derecho del ratón y accede a propiedades (Properties). A continuación, seleccionamos Ejecutar (run) y en la parte de la derecha, opción Main class, pulsaremos el botón Examinar (Browser) para seleccionar la clase principal.

4.- Contenedores.

Un **formulario** es una ventana que dispone de tres botones para m1rnm1zarse, max1m1zarse o cerrarse, una barra de título y está delimitado por unos bordes. Es la base para crear una aplicación de escritorio. Sobre él se añadirán controles o componentes, sencillos como botones o cajas de texto o más complejos, como barras de menú o rejillas de datos, que le dan funcionalidad.



Una aplicación de escritorio de NetBeans se compone de una serie de formularios. Para crear un formulario, tendrás que usar un **contenedor** Java que es un componente que permite incluir otros componentes incluidos otros contenedores que se usarán para distribuir los controles. Por eso se dice que los contenedores forman una **estructura jerárquica.**

Un formulario está formado por un contenedor especial que se llama contenedor de nivel superior. Este tipo incluye un panel de contenido {contentpane} que permite añadir otros componentes, incluidos otros contenedores que se utilicen facilitar la distribución de elementos.

Como contenedor de nivel superior de un formulario puedes elegir entre una ventana {JFrame}, un diálogo

{JDialog} o un applet {JApplet}, según la necesidad. Todos estos componentes derivan, en la jerarquía de clases de java, de Window que representa una ventana típica.

- **Ventana {JFrame}:** es un formulario con título, los botones para maximizar, minimizar o cerrar y borde. Aparece un icono por defecto en forma de taza de café que puedes modificar, y puede contener una barra de menú.
- **Dialogo {JDialog}:** formularios que se suelen usar para solicitar información al usuario. Su principal característica es que pueden ser modales o no, una ventana modal recibe todas las entradas de usuario e impide que se active otra ventana.
- **Applet {JApplet}:** ventana que ejecuta una aplicación Java en el contexto de una página web.

En una aplicación de escritorio se suele crear una ventana principal que sea de tipo JFrame y si necesitamos ventanas secundarias utilizaremos otras ventanas o diálogos.

4.1.- Contenedores secundarios.

También se interpretan como diálogos los siguientes componentes:

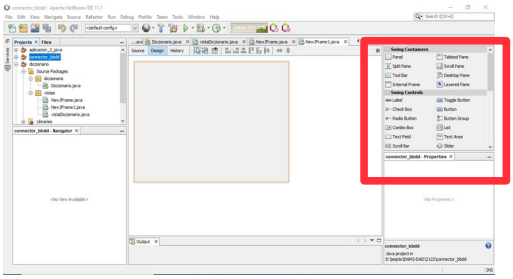
- **Panel de opciones (JOptionPane):** genera ventanas con botones para responder cuestiones con respuestas del tipo si-no, aceptar-cancelar, aceptar, etc.

- **Selector de archivos (JFileChooser):** permite seleccionar un archivo del sistema de archivos del equipo donde se ejecuta la aplicación .
- **Selector de colores (JColorChooser):** permite seleccionar entre un conjunto de colores y devolverlo usando el código adecuado.

Puedes usar otro tipo de contenedores para distribuir el resto de los controles que se incluyen en la ventana principales, entre los más habituales tienes:

- **Paneles (JPanel):** representa un contenedor intermedio, cuya función principal es la de colocar controles.
- **Barra de menús (JMenu):** permite la creación de menús complejos de opciones.
- **Barra de herramientas (JToolBar):** se utiliza para contener iconos de acceso a las opciones de la aplicación.
- **Pestañas (JTabbedPane):** tipo particular de panel que permite la distribución de elementos en pestañas o tarjetas.
- **Paneles deslizables (JScrollPane):** tipo especial de panel que permite desplazar sus contenidos de manera automática .
- **Ventanas internas (JInternalFrame):** ventanas hijas que no pueden rebasar los límites de la ventana padre donde se han creado. Se usan en aplicaciones que tienes varios documentos abiertos simultáneamente .
- **Paneles divididos (JSplitPane):** permite visualizar dos componentes, uno a cada lado, asignando espacio dinámicamente a cada uno.

5.- Componentes de la interfaz.



Los componentes o controles gráficos de un formulario, son elementos gráficos que se anidan en los contenedores para formar aplicaciones. Se utiliza para mostrar información, como etiquetas o imágenes, listas (componentes pasivos) o árboles, pero, sobre todo, para recabar información del usuario, como cuadros de texto, botones, o listas de selección (componentes activos).

Un componente se reconoce por su clase, que define su aspecto y funcionalidad y por su **nombre** que lo identifica dentro de la aplicación. Puesto que es un objeto, según la clase a la que pertenezca tendrá, una serie de propiedades que podremos modificar para adaptar el componente, por ejemplo, el texto mostrado, o el color.

La **colocación de componentes** en el formulario se rige por unas reglas, denominadas Layout, que establecen el orden y la posición en la que deben ser mostrados, pudiendo ser en torno a los límites del formulario (norte, sur, este y oeste), en forma de rejilla, o fluidos, uno tras otro, en una o varias filas.

Cuando un componente es susceptible de interactuar con el usuario se gestiona mediante lo que se conoce como **manejo de eventos,** una acción sobre el componente que debe provocar una respuesta se conoce como evento, la gestión de la respuesta se realiza a través de los manejadores de eventos, que son unas funciones específicas que se asocian a un elemento del componente denominado escuchador, en las que se programa la acción a realizar.

5.1.- Añadir y eliminar componentes de la interfaz.

Los componentes se pueden añadir desde la paleta, que, si recordamos suele anclarse a la derecha de la interfaz del IDE NetBeans.

A la derecha tienes la lista de controles para añadir a una interfaz en NetBeans. Están organizados en las siguientes categorías:

- **Contenedores swing: s**on secundarios en la jerarquía de contenedores y se usan para distribuir y organizar el resto de controles.
- **Controles Swing:** básicos para crear una interfaz útil para comunicarse con el usuario y mostrar o solicitar información.
- **Menús Swing:** incluyen los controles necesarios para crear menús de aplicación complejos, con varios bloques, elementos activos e inactivos, etc y menús contextuales (Popup Menu)
- **Ventanas Swing:** permiten añadir a la aplicación ventanas (JFrame), diálogos (JDialog), selectores de ficheros y colores (JFileChooser y JColorChooser) y paneles de opciones (JOptionPane) para crear diálogos que se contestan con Sí/No.

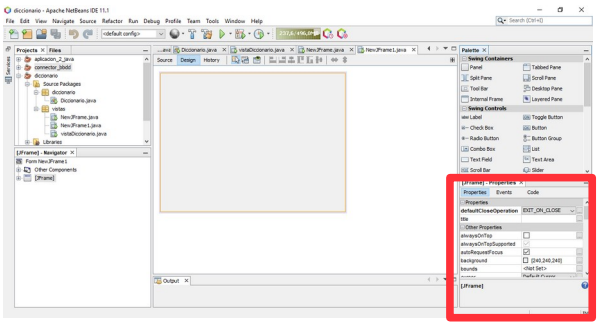
Para añadir componentes a la interfaz, seleccionaremos el control en la paleta y pinchando sobre la interfaz que se está construyendo. El control aparece en la interfaz con su aspecto por defecto que puedes modificar. Si arrastras el control se moverá sobre la superficie de su contenedor y si haces clic sobre una esquina y desplazas el ratón lo cambiarás de tamaño.

Al colocar un control sobre un formulario aparecen unas guías que te permiten colocarlo con más facilidad, esto será así mientras que tengas activa la opción **diseño libre**, lo puedes comprobar en el inspector haciendo clic con el botón secundario en el nodo raíz de la interfaz y seleccionando activar gestor de distribución. Si **mueves** un control estas guías te permitirán relacionarlo con otros componentes para que lo puedas alinear mejor.

Conforme vas colocando controles en el panel del formulario éstos aparecen reflejados, además, en el **Inspector**, de forma que los seleccionas tanto haciendo clic sobre ellos como sobre su nombre. Trabajar con el inspector facilita colocar controles dentro de contenedores porque admite operaciones de arrastrar y soltar.

Para **eliminar** un control basta con seleccionarlo {de cualquiera de las formas descritas) y pulsar la tecla Supr, o bien seleccionar la opción Suprimir del menú contextual (el menú contextual aparece cuando pulsamos el botón derecho del ratón).

5.2.- Modificación de propiedades.



Una vez que has colocado un control en el formulario puedes modificar sus propiedades para adaptarlo a tu interfaz. Con el control seleccionado accedes a sus propiedades en el panel propiedades, que lógicamente, será diferente para cada tipo de control.

Se suele comenzar por modificar el **nombre** del control para localizarlo con más facilidad en el código de la clase que genera el formulario. Para hacerlo puedes sacar el menú contextual del control en el Inspector y seleccionar cambiar nombre de Ja variable...

Otra propiedad muy común es el **ToolTipText**, es un pequeño recuadro de color amarillo, normalmente, en el que aparece una breve descripción de para qué sirve el control. Lo puedes establecer en la propiedad **ToolTipText**.

5.3.- Añadir funcionalidad desde NetBeans.

Creamos interfaces para permitir que el usuario pueda interactuar con la aplicación, pero siempre será necesario añadir código para darles la funcionalidad para la que ha sido creadas. Cuando usamos un entorno integrado como NetBeans parte de este código se genera de forma automática facilitando en gran medida el trabajo del desarrollador o desarrolladora, pero tendrás que abrir este código y modificar algunas cosas para que el formulario realice las tareas para las que ha sido diseñado.

5.4.- Ubicación y alineamiento de los componentes.

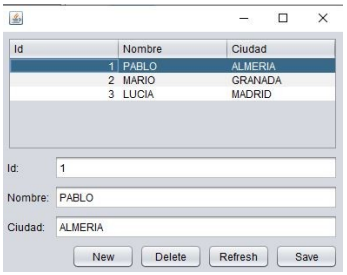
Internamente la herramienta emplea el mecanismo de Java para disponer los elementos llamado Layout, distribución o diseño. Swing dispone de ocho tipos de distribuciones:

- **BoderLayout**: aloja los componentes en los límites del formulario, por lo que cuando los colocamos debemos indicar si van al norte, sur este u oeste.
- **GridLayout**: Diseña mediante una rejilla, en la que los componentes se organizan por filas y columnas.
- **GridBagLayout**: semejante a GridLayout, pero permite a un componente que ocupe más de una celda.
- **CardLayout**: diseño por paneles. Permite la colocación de distintos componentes en momentos distintos de la ejecución.
- **BoxLayout**: diseño en caja. Coloca los componentes en una fila o columna ajustándose al espacio que haya.
- **FlowLayout**: diseña alojando los componentes de izquierda a derecha mientras quede espacio, si no queda pasa a la fila siguiente.
- **GroupLayout**: se creó para ser utilizado en herramientas de diseño gráfico de interfaces. Trabaja por separado la distribución vertical y horizontal para definir exactamente el posicionamiento de los componentes. Se utiliza en NetBeans.
- **SpringLayout**: es muy flexible y se usa también para herramientas de diseño gráfico de interfaces. En este caso se especifican las relaciones entre los límites de los componentes bajo su control.

Programar el diseño de un formulario es una de las tareas más arduas en Java, si bien está ampliamente superado gracias al uso de IDEs que facilitan la colocación de componentes a golpe de ratón y sin necesidad de escribir código. Por ejemplo, NetBeans, usa el diseño **GroupLayout**..

5.5.- Enlace de componentes a orígenes de datos.

Los pasos a dar para conseguir que una aplicación java, utilizando NetBeans, incorpore un formulario basándose en los datos almacenados en una tabla de una base de datos son:



1. Nos aseguramos de tener instalado mysql y que se encuentra en ejecución. A continuación, crearemos la base de datos. La base de datos que vamos a utilizar se llamará Agenda. Puedes importar la estructura de la base

de datos junto con sus datos utilizando el fichero que encontrarás en apartado Recursos que aparece al final de la página (BD_agenda.sql)

Recomendación: Antes de comenzar a crear el proyecto en NetBeans, accede al apartado Services o Prestaciones y comprueba que podemos ver que aparece Servidor Mysql y que podemos acceder a la base de datos Agenda. En el caso de que no aparezca debes de registrar servidor Mysql e iniciarlo (para ello accedemos al menú contextual de la opción Base de datos y realizamos las dos acciones.)

2. Crearemos un nuevo proyecto NetBeans de tipo Java Application que se llame Agenda. No crearemos clase principal al crearlo. Añadimos un paquete para almacenar todas las clases que se van a generar de forma automática. Se va a llamar sources y lo vamos a crear dentro de Paquete de fuentes. Añadimos un Formulario de ejemplo/detalle que podemos encontrarnos dentro de la categoría Formularios de interfaz gráfica de Swing. Le asignaremos el nombre Contactos.

3. A continuación, nos pedirá el acceso a la base de datos. En la conexión con la base de datos marcamos la cadena de conexión jdbc:mysl..... que nos aparecerá en el desplegable. Si no nos apareciera nuestra conexión con MySQL, este desplegable nos permite crear una conexión nueva. En el apartado Tabla de base de datos, seleccionamos la tabla Contactos y en el apartado Campos a incluir nos aseguramos de tener incluido las columnas que queremos utilizar en el formulario.

4. En el apartado Crear área de detalle como seleccionaremos la opción cuadro de texto.

5. Finaliza el asistente pulsando Terminar, en el formulario Agenda aparecerán una tabla para mostrar el resultado de la consulta.

Si seleccionas la tabla que ha aparecido en el formulario y observas el Inspector verás que se llama **masterTable** (Jtable). Esta tabla en concreto tiene tres columnas y se le asigna contenido a partir de un elemento, que también puedes ver más abajo en el Inspector llamado **list** (JList) que se rellena a partir de una consulta llamada Query (Jquery), en cuyas propiedades aparece la consulta ejecutada que es **Select c from contactos c.** Haz clic con el botón secundario sobre la tabla >> Enlazar >> Elements, verás cómo enlaza con la lista y como se obtienen de ella los tres campos de la tabla.

Por otra parte, se han añadido tres etiquetas y tres campos de texto al formulario, los campos de texto enlazan (botón secundario >> Enlazar >> Text) con la tabla maestra en el enlazado de fuente de donde se saca el valor del campo del registro actual mediante la expresión **${SelectedElement.nombre_campo}** de esta manera cada vez que se cambia el registro activo de la tabla se modifica el contenido de los campos de texto.

6.- Asociación de acciones a eventos.

En las aplicaciones que utilizan una interfaz gráfica de usuario (GUI), a diferencia de la ejecución de un programa de consola, donde el orden de ejecución de las instrucciones dependerá del método Main, el orden de ejecución dependerá de la interacción que realiza el usuario o usuaria sobre la aplicación.

Durante el diseño de la aplicación se deberá considerar qué acciones deseamos que realice nuestra aplicación, cuándo debe realizarlas, y definir los manejadores de eventos que serán invocados de manera automática cuando sobre la aplicación se produzca el evento.

En el lenguaje Java se gestiona el modelo de eventos de la siguiente forma: los objetos sobre los que se producen los eventos (event sources) "registran" los objetos que habrán de gestionarlos (event listeners), para lo cual los event listeners habrán de disponer de los métodos adecuados. Estos métodos se llamarán automáticamente cuando se produzca el evento. La forma de garantizar que los event listeners disponen de los métodos apropiados para gestionar los eventos es obligarles a implementar una determinada interfaz Listener.

Las interfaces Listener se corresponden con los tipos de eventos que se pueden producir. En los apartados siguientes se verán con más detalle los componentes que pueden recibir eventos, los distintos tipos de eventos y los métodos de las interfaces Listener que hay que definir para gestionarlos. En este punto es muy importante ser capaz de buscar la información correspondiente en la documentación de Java.

Cuando se está diseñando una aplicación, una vez que se ha añadido los diferentes componentes de la interfaz, se procede a definir los objetos Listener, que son los objetos que se encargan de responder a los eventos, para cada evento que se quiera soportar. Una vez definidos los objetos Listener, se procede a implementar los métodos de las interfaces Listener que se vayan a hacer cargo de la gestión de eventos.

7.- Diálogos modales y no modales.

Cuando se diseña una aplicación de interfaz gráfica, el elemento básico es el diálogo o ventana. El diálogo es un área visual que contiene los elementos de interfaz de usuario, tales como menús, botones, listas, etc. mostrando la aplicación y permitiendo la entrada de datos.

Los diálogos se representan casi siempre como objetos de dos dimensiones colocados en el escritorio. La mayoría de ellos pueden ser redimensionados, movidos, ocultados, restaurados, etc.

Existen dos modalidades de diseñar diálogos. La modalidad se refiere a la forma de mantener el foco que va a tener el diálogo con respecto a los demás diálogos.

Un diálogo será no modal, si una vez que se encuentra activo permite alternar el foco a cualquier otro diálogo que se encuentre abierto en el sistema o dentro de la propia aplicación. Normalmente, la mayoría de los diálogos son de este tipo. Dentro de los diálogos

modales nos podemos encontrar los modales respecto a una aplicación o los modales respecto al sistema.

Los modales respecto a una aplicación permiten alternar el foco a otros diálogos del sistema, pero no al diálogo que le da origen (diálogo padre) hasta que se produzca una determinada acción sobre ella. Típicamente, son diálogos modales de aplicación, los que se implementan para confirmar una acción del usuario.

Un diálogo modal respecto al sistema no va ceder el foco a ninguna otra aplicación hasta que se produzca una determinada acción sobre él. No suelen ser muy habituales, salvo para cuando se quiere gestionar un evento a nivel de sistema, que requiera la atención inmediata del usuario. Un ejemplo podría ser un diálogo que permita apagar el equipo.

7.1.- Diálogos modales.

En una aplicación de interfaz gráfica en Java es necesario definir un objeto de la clase Frame. Este objeto va a ser el diálogo padre de toda la aplicación, derivando de él el resto de diálogos que queramos añadir a la aplicación. Para añadir un diálogo modal, Java ya permitía la creación de diálogos modales a través del constructor de la clase JDialog como hemos comentado en el punto anterior.

Para crear diálogos modales, en el constructor del diálogo establecemos el parámetro "modal" a true, de forma que este diálogo creado mantiene el foco, impidiendo que pueda ser tomado por cualquier otro, dentro de la aplicación, de forma que no podemos seguir ejecutando la aplicación hasta cerrarlo.

```
package mipaquete;
import javax.swing.*;
public class Ventana {
  public static void main(String[] args) {
   JFrame ventana = new JFrame("");
   ventana.setAlwaysOnTop(true);
   ventana.setSize(300,300);
   ventana.setVisible(true);
   JDialog dialogo_Modal = new JDialog(ventana, "Dialogo Modal", true);
   dialogo_Modal.setSize(300,300);
   dialogo_Modal.setLocationRelativeTo(null);
   dialogo_Modal.setVisible(true);
  }
}
```

Se puede definir la modalidad de un diálogo utilizando dos clases estáticas dentro de la clase Dialog. Las clases son static class ModalityType y static class ModalExclusionType. Las dos clases incorporan una enumeración que indica el nivel de bloqueo que pueden generar.

La clase static class ModalityType sirve para definir el tipo de bloqueo, o también llamado alcance del bloqueo en una aplicación, su enumeración contiene:

- **APPLICATION_MODAL**: bloquea todas las ventanas de la aplicación, excepto las ventanas que se deriven de ella.
- **DOCUMENT_MODAL**: bloquea la ventana padre de la ventana y su jerarquía superior.
- **MODELESS**: no bloquea ninguna ventana.
- **TOOLKIT_MODAL**: bloquea las ventanas de nivel superior que corren en el mismo TOOLKIT.

Podemos excluir del bloqueo a una ventana o diálogo utilizando la clase ModalExclusionType, esta puede excluir del bloqueo según alguna de estas opciones de su enumeración:

- **APPLICATION_EXCLUDE:** La ventana que utiliza este valor de enumeración no será bloqueada por ningún diálogo de la aplicación.
- **NO_EXCLUDE:** Indica que la ventana no estará excluida del bloqueo si este ocurriera.
- **TOOLKIT_EXCLUDE:** Indica que no se bloqueará esta ventana si se llamase a APPLICATION_MODAL o TOOLKIT_MODAL.

7.2.- Diálogos no modales.

Dentro de una aplicación de interfaz gráfico, nos podemos encontrar con aplicaciones que presentan varias ventanas o diálogos abiertos de manera simultánea. Diremos que los diálogos son no modales, si podemos pasar el foco de un diálogo a otro sin ningún tipo de restricción.

Un ejemplo de aplicación GUI que presenta formularios no modales es el propio entorno de desarrollo NetBeans, donde podemos pasar de una ventana a otra sin ningún problema.

Un diálogo no modal permite cambiar el foco a cualquier otro diálogo o ventana abiertos en el sistema.

Para definir diálogos no modales en Java, el código que se utiliza es el siguiente:

```
package mipaquete;
import javax.swing.*;
public class Ventana {
  public static void main(String[] args) {
```

```
        JFrame ventana = new JFrame("");
        ventana.setAlwaysOnTop(true);
        ventana.setSize(300,300);
        ventana.setVisible(true);
        JDialog dialogoNoModal = new JDialog(ventana,"Dialogo No Modal");
        dialogoNoModal.setSize(300,300);
        dialogoNoModal.setLocationRelativeTo(null);
        dialogoNoModal.setVisible(true);
    }
}
```

8.- Edición de código generado por herramientas de diseño.

La implementación de una aplicación de interfaz gráfica (GUI requiere la definición de muchos objetos para la interfaz, establecer sus propiedades, conocer los eventos que se quieren controlar para poder crear los manejadores de eventos, etc. Si la aplicación es simple, puede resultar asumible la implementación de esa forma, pero si la aplicación tiene cierta envergadura, sería un trabajo arduo y lento.

En la actualidad no se concibe la implementación de una aplicación GUI sin el uso de un entorno de desarrollado que, de manera rápida y visual, nos permita crear los diálogos de la aplicación, añadir aquellos objetos que deseemos y establecer de forma gráfica y rápida el aspecto que nos interese.

Con los IDE, como por ejemplo NetBeans, podemos crear la interfaz de manera rápida, y sin tener que implementar una gran cantidad de líneas de código, que el entorno realiza por nosotros.

A pesar de todas las facilidades que nos ofrecen los IDE, siempre es necesario modificar determinas aspectos del código. Para ello podemos modificar con la ventana de código.

El IDE automáticamente genera bloques de código cuando realizas el diseño de una ventana con las herramientas gráficas en "Vista Diseño". Este código se muestra enmarcado dentro del fichero .java, pero puede ser modificado. Se puede modificar la manera de inicializar los componentes, los formularios, las propiedades de los componentes editándolos en la ventana "Fuente" que nos muestra el código fuente generado a respuesta a tu diseño gráfico. Además, puedes escribir código personalizado y específico donde consideres necesario.

Para modificar un componente de la ventana, se puede actuar de la siguiente forma:

1. En la ventana Navegador, seleccionas el componente que al que quieres editar su código de inicialización.
2. Haz clic en el botón Código, que aparece en la ventana Propiedades, para ver el código de las propiedades.

3. Selecciona la propiedad que desees editar y añade el valor que quieras.

El IDE actualizará el componente seleccionado en el código que había generado con el nuevo valor.

Para modificar la inicialización de código generado para una propiedad de un componente se debes actuar de la siguiente forma:

1. Selecciona el componente en la ventana o panel Navegador.
2. Haz clic en el botón Propiedades en la ventana de propiedades.
3. Selecciona la propiedad que quieras modificar en el código de inicialización.
4. Ha clic en el botón (...) para llegar a la ventana de diálogo Editor de Propiedades.
5. Establece el valor de que desees que tenga la propiedad en cuestión.

Para modificar de manera libre el código generado por el IDE, simplemente puedes seleccionar el botón "Fuente" (Sources) que aparece en la barra de herramientas del IDE, y podrás modificar directamente cualquier parte del código y adaptarlo a tus necesidades.

9.- Clases, propiedades, métodos.

Para la implementación de Interfaces Gráficas de Usuario en Java, JavaSoft ha creado un conjunto de clases que son agradables desde el punto de vista visual y fáciles de utilizar para el programador. Esta colección de clases son las Java Foundation Classes (JFC), que en la plataforma Java 2 están constituidas por cinco grupos de clases: AWT, Java 2D, accesibilidad, arrastrar y soltar y swing.

- AWT. Bibliotecas de clases Java para el desarrollo de Interfaces gráficas de usuario.
- Java 2D. Es un conjunto de clases gráficas baja licencia IBM/Taligent.
- Accesibilidad, proporciona clases para facilitar el uso de ordenadores y tecnología informática a personas con deficiencias visuales como lupas de pantallas, etc.
- Arrastrar y soltar (Drag and Drop), son clases en la que se soporta los JavaBeans.
- Swing. Es la parte más importante para el desarrollo de aplicaciones de interfaz gráfica. Swing proporciona un conjunto de componentes muy bien descritos y especificados, de forma que su presentación visual es independiente de la plataforma donde se ejecuta la aplicación que utilice sus clases. Swing extiende AWT añadiendo un conjunto de componentes, **JComponents**, y sus clases de soporte.

Actualmente, swing ha desplazado a AWT debido a que los componentes de swing, al estar escritos en Java, ofrecen mayor funcionalidad, e independiente de la plataforma.

9.1.- Clases

En el desarrollo de interfaces gráficas de usuario basada en el lenguaje Java, actualmente se utilizan los componentes swing.

Los componentes swing son objetos de clases derivadas de la clase **`Jcomponent`** que deriva de la clase **`java.awt.Component`**. Para crear interfaces gráficas de usuario, swing combina los componentes de la clase **`Jcomponent`** en contenedores de nivel alto **`Jwindow`**, **`Jframe`**, **`Jdialog`** y **`JApplet`**.

- **`Jwindow`** es una ventana sin barra de título y sin botones.
- **`Jframe`** es una ventana con barra de título y con los botones que permiten su manipulación.
- **`Jdialog`** permite visualizar una cuadro de diálogo.
- **`Japplet`** permite crear un **applet swing.**

Para dar funcionalidad a las ventanas, el swing proporciona un conjunto de componentes que derivan de la clase JComponent, los más utilizados son:

- **`JComponent`**: Clase base para los componentes swing.
  - **`AbstractButton`**: Define el comportamiento común de los botones y los menús.
    - **`JButton`**. Botón.
    - **`JMenuItem`**. Elemento de un menú.
      - **`JCheckBoxMenuItem`**: Elemento del menú que se puede seleccionar o deseleccionar.
      - **`JMenu`**: Menú de una barra de menús.
      - **`JRadioButtonMenuItem`**: Elemento que forma parte de un grupo de elementos de menú.
    - **`JToggleButton`**: Botón de estados.
      - **`JCheckBox`**. Casilla de verificación.
      - **`JRadioButton`**: Botón de opción.
  - **`JColorChooser`**: Diálogo para seleccionar colores.
  - **`JComboBox`**: Combinación de caja de texto y lista desplegable.
  - **`JLabel`**: Etiqueta.
  - **`JList`**: Lista desplegable.
  - **`JMenuBar`**: Barra de menús.
  - **`JOptionPane`**: Componente que facilita la visualización de un cuadro de diálogo.
  - **`JPanel`**: Contenedor genérico.
  - **`JPopupMenu`**: Menú que aparece cuando se selecciona un elemento de una barra de menús.
  - **`JProgressBar`**: Barra de progreso.
  - **`JScrollBar`**: Barra de desplazamiento.

- **`JScrollPane`**: Área de trabajo con barras de desplazamiento.
- **`JSeparator`**: Separador para colocar entre dos elementos del menú.
- **`JSlider`**: Permite seleccionar un valor dentro de un intervalo que define.
- **`JTableHeader`**: Se utiliza para manejar la cabecera de una tabla.
- **`JTextComponent`**: Clase base para los componentes de texto.
  - **`JEditorPane`**: Edita diferentes tipos de contenido.
    - **`JTextPane`**: Edita texto con formato, incluyendo imágenes.
  - **`JTextArea`**: Caja de texto multilínea.
  - **`JTextField`**: Caja de texto de una línea.
    - **`JPasswordField`**: Se usa para introducir contraseñas.
- **`JToolbar`**: Barra de herramientas.

9.2.- Propiedades

Las propiedades de los componentes nos van a permitir adaptar su comportamiento y apariencia.

Las propiedades más importantes que presentan la mayoría de los componentes swing son las siguientes:

- **background**: Determina el color de fondo del componente.
- **font**: Establece el tipo de fuente que va a mostrar el componente.
- **foreground**: Establece el color de la fuente que muestra el componente.
- **horizontalAlignment**: Establece la alineación del texto dentro del componente en el eje X.
- **icon**: Indica si el componente muestra o no un icono, y cual sería.
- **labelFor**: Indicaría si el componente es etiquetable.
- **text**: Muestra el texto que indica la propiedad en componentes como caja de texto o etiquetas.
- **toolTipText**: Con esta propiedad definimos el texto que aparece como tool tip.
- **verticalAlignment**: Establece la alineación del texto dentro del componente en el eje Y.
- **aligmentX**: Alineamiento horizontal preestablecido para el componente.
- **aligmentY**: Alineamiento vertical preestablecido para el componente.
- **autoscrolls**: Determina si el componente puede realizar scroll de forma automática cuando se arrastra el ratón.
- **border**: Establece el tipo de borde que va presentar el componente.

- **componentPopupMenu**: Estable el menú contextual que se muestra en este componente.
- **cursor**: Establece el tipo de cursor que se va a mostrar cuando el curso entre en el componente.
- **disableIcon**: Establece el icono a mostrar si el componente no está activo.
- **enabled**: Nos indica si el componente está o no activo.
- **focusable**: Establece si el componente puede o no, recibir el foco.
- **horizontalTextPosition**: Establece la posición horizontal del texto del componente, en relación con su imagen.
- **iconTextGap**: Si el componente tiene activo el texto y el icono, con esta propiedad se establece la distancia entre ellos.
- **inheritsPopupMenu**: Indica si el menú contextual se hereda o no.
- **inputVerifier**: Establece el componente de verificación para este componente.
- **maximunsize**: Establece el tamaño máximo del componente.
- **minimunsize**: Establece el tamaño mínimo del componente.
- **name**: Establece el nombre del componente.
- **opaque**: Modifica la opacidad del componente.
- **preferredSize**: Tamaño predefinido para este componente.
- **verifyInputWhenFocusTarget**: Si el componente es verificado antes de recibir el foco.
- **verticalTextPosition**: Establece la posición vertical del texto del componente, en relación con su imagen.

Las propiedades que hemos enumerado anteriormente, son las más habituales que te puedes encontrar en los componentes de las swing de Java. Para modificar los valores de las propiedades dispones de la ventana de propiedades del IDE, o bien puedes hacerlo desde el código fuente Java. Si deseas modificar el valor de una propiedad desde el código fuente, el IDE te va a mostrar una lista de todas las propiedades disponibles para el componente en cuestión, una vez que escribas el nombre del objeto y un punto.

9.3.- Métodos

Cada componente que forma parte de una aplicación GUI utilizando Java, dispone de un conjunto completo de métodos, que nos permite comunicarnos con el componente y modificar su aspecto o comportamiento. A continuación, se va a mostrar una lista de los componentes más importantes de swing Java, junto con sus métodos más destacados:

- `JFrame`. Los métodos más importantes son: `getTitle()`, `setBounds(x,yx,w,h)`, `setLocation`, `setMaximumSize(w,h)`, `setMinimumSize(w,h)`, `setPreferredSize(w,h)`, `setResizable(bool)`, `setSize(w,h)`, `setTitle(str)` y `setVisible(bool)`.

- **`JPanel`**. El panel de contenido dispone del método **`add()`**, para añadir componentes al panel.
- **`JLabel`**: Las etiquetas tienen como métodos más importantes: **`setText()`** que permite modificar el texto, **`setVerticalAlignment()`** para modificar la alineación vertical, etc.
- **`JButton`**: Los botones presentan entre otros métodos: **`setEnabled(bool)`** que permite activar o desactivar el botón, **`isEnabled()`** que permite comprobar si está o no activo, **`setMnemonic(char)`** que permite asociar una tecla al botón, etc.
- **`JprogressBar`**, **`JScrollBar`**. Estos componentes disponen de los siguientes métodos: **`setExtent()`**, **`setMaximum()`**, **`setMinimum()`**, **`setValue()`**, **`getValueIsAdjusting()`** y **`setOrientation()`**.
- **`JSlider`**: Este componente tiene como métodos más destacados: **`setPaintTicks(bool)`**, **`setPaintLabels(bool)`**. **`setMajorTickSpacing(int)`** y **`setMinorTickSpacing(int)`** para determinar los valores del intervalo, **`setSnapToTicks(true)`** y **`setInverted(bool)`**.
- **`JRadioButton`**. Este componente tiene como método principal **`isSelected()`** que devuelve el estado del mismo.
- **`JList`**. En las listas desplegables destacan los métodos siguientes para editar los elementos de la lista: **`addElement(objName)`**, **`elementAt(index)`**, **`removeElement(objName)`**, **`removeAllElements()`** y **`removeElementAt(idx)`**. Para comprobar el estado de la lista se usan los métodos **`contains(objName)`**, **`indexOf(name)`** y **`size()`**. Para convertir la lista en array se usa **`copyInto(array)`**.
- **`JComboBox`**. Dispone de diferentes métodos, destacando **`setEditable(true)`** que permite editar los items del combo box. Para recuperar los ítems seleccionados se usa **`getSelectedItem()`** y **`getSelectedIndex()`**. Otros métodos útiles son **`getItemAt(idx)`**, **`getItemCount()`**, **`setSelectedItem(obj)`** y **`setMaximumRowCount(x)`**.
- **`JtextPane`** y **`JTextArea`**. Como métodos más útiles de estos componentes destacan **`getText()`**, **`setText()`**, **`append(str)`**, **`insert(str,pos)`**, **`replace(str,beg,end)`**, **`setEditable(bool)`**, **`setLineWrap(bool)`** y **`setWrapStyleWord(bool)`**.
- **`JMenuItem`**. Para añadir separadores se usa el método **`addSeparator()`** o **`insertSeparator(posn)`**. Los Menú ítems se pueden activar o desactivar con **`setEnabled(bool)`** y comprobar si están activos con **`isEnabled()`**. Para añadir teclas de acceso rápido se usa el método **`setMnemonic(char)`**.

Cada componente dispone de muchos más métodos que puedes utilizar en tus aplicaciones, utilizando un IDE como NetBeans, una vez que hayas definido un objeto, por ejemplo, el objeto **`etSaludo`** de la clase **`JLabel`**, cuando invoques **`etSaludo`** y escribas a continuación un punto, automáticamente te aparecerá una lista con todos los métodos que puedes usar para ese objeto, junto con una breve explicación de su uso.

10.- Eventos, escuchadores.

En un entorno GUI, cuando se produce un movimiento de ratón, se pulsa una tecla, se cierra una ventana, etc. se produce un evento. El evento es recibido por un objeto, como puede ser un botón, una caja de texto, etc. El objeto que recibe el evento, para poder responder a él, debe implementar la interfaz apropiada (event handler) y registrarla como un escuchador (event listener) en el componente GUII (event source u objeto que va recibir el evento) apropiado.

Los eventos se agrupan en función al componente que lo produce o a la acción que lo provoca.

El evento tiene los siguientes elementos:

- La **fuente del evento** (event source): Es el componente que origina el evento.
- El **escuchador** (event listener): es el encargado de atrapar o escuchar el evento.
- El **manejador** del evento (event handler), es el método que permite implementar la interfaz.

El manejador del evento recibe un objeto evento (ActionEvent) que contiene información sobre el evento que se ha disparado, cuando esto sucede, el manejador del evento descifra el evento con dicho objeto y procesa lo solicitado por el usuario o usuaria.

Un componente de una interfaz gráfica dispara o activa los manejadores (event handler) dependiendo del tipo de evento que ha ocurrido. El componente puede tener registrado más de un manejador que maneja el mismo tipo de evento. Un evento puede ser escuchado por más de un manejador de eventos.

El manejador del evento requiere que en la declaración de la clase que maneja el evento (event handler), se indique la interfaz correspondiente al evento (XListener, donde X es el tipo de evento a escuchar). La clase puede implementar más de un Xlistener. Si la clase no implementa la interfaz puede ser que extienda a una clase que si la implementa:

```
public class miClase implements ActionListener{...}
```

En los componentes que son la fuente del evento (event source) se registra una instancia del manejador del evento (event handler), como un observador o escucha del tipo de eventos que maneja (`XListener`). Es decir, se le dice al componente que va a escuchar los eventos del tipo del manejador

```
componente.addActionListener( instancia_de_miClaseListener);
```

En la clase que maneja el evento (event handler) se deben implementar los métodos de la interfaz XListener que descifren el evento (XEvent) y lo procesen.

```
public void actionPerformed(ActionEvent e) {
    ...//Código que reaccione a la acción
```

```
}
```

10.1.- Escuchadores.

Un escuchador (listener) es el objeto de la clase que implementa la interfaz de escucha de un evento y que contiene el método de respuesta al propio evento. Cuando se produce un determinado evento dentro del aplicación GUI, si se desea responder a esos eventos, la aplicación deberá implementar una serie de métodos que se ejecuten de forma automática cuando se produzca esos eventos. Los métodos que se van a implementar para responder a los diferentes eventos dentro de una aplicación de interfaz gráfica, se definen en unas interfaces denominadas interfaces de escucha.

Cada interfaz de escucha contiene los métodos para gestionar un determinado grupo de eventos. La interfaz de escucha **`WindowsListener`** define el formato de los métodos para la gestión de los eventos de ventana, como pueden ser abrir la ventana, cerrarla, maximizarla, minimizarla etc. Algunos de los métodos de **`WindowsListener`** son:

- **`public void windowActivated(WindowEvent e)`**: Invocado cuando la ventana es la ventana activa.
- **`public void windowDesactivated(WindowEvent e)`**: Invocado cuando la ventana deja de ser la ventana activa.
- **`public void windowClosing(WindowEvent e)`**: Cuando la ventana se ha cerrado.
- **`public void windowClosed(WindowEvent e)`**: Cuando la ventana se minimiza.
- **`public void windowIconified(WindowEvent e)`**: Cuando la ventana se restablece.
- **`public void windowDeiconified(WindowEvent e)`**: La primera vez que la ventana se minimiza.
- **`public void windowOpened(WindowEvent e)`**: La primera vez que la ventana se hace visible.

Cada interfaz de escucha contiene sus propios métodos para la gestión de eventos. Ejemplos de eventos que son generados por componentes de swing son:

- **`ActionListener`**: Captura cierto tipo de acción realizada sobre ciertos componentes. Por ejemplo, pulsar un botón, seleccionar un elemento en una lista desplegable o una opción en un menú.
- **`ChangeListener`**: Registra los cambios sobre ciertos componentes.
- **`ItemListener`**: Recoge el cambio de estado en un componente tipo listas desplegables.
- **`MouseListener`**: Eventos producidos por la manipulación del ratón.

- **`MouseMotionListener`**: Movimiento del ratón sobre un componente.
- **`ComponentListener`**: Visibilidad de componentes.
- **`FocusListener`**: Captura eventos relacionados con la recepción o pérdida del foco.
- **`KeyListener`**: Captura pulsaciones del ratón sobre el componente activo.
- **`ListSelectionListener`**: Selección de elementos dentro de una lista.

Para responder a un evento dentro de una aplicación, deberás definir una clase que interprete la interfaz de escucha correspondiente al tipo de evento que quieras gestionar. A los objetos de esa clase de escucha se les denomina escuchadores.