DESARROLLO DE INTERFACES

UNIDAD 1. CONFECCIÓN DE INTERFACES DE USUARIO

ÍNDICE DE CONTENIDOS

INTRODUCCIÓN

La confección de interfaces de usuario es un aspecto fundamental en el desarrollo de aplicaciones, puesto que estas interfaces impactan directamente en la experiencia del usuario, influyendo en su satisfacción y efectividad al interactuar con el software. Para facilitar este proceso, existe una amplia gama de librerías de componentes que se adaptan a diversos sistemas operativos y lenguajes de programación. Estas librerías proporcionan herramientas y elementos visuales predefinidos que los desarrolladores pueden utilizar para crear interfaces de forma más eficiente. Por ejemplo, librerías como React para JavaScript, Xamarin para la creación de aplicaciones móviles, y JavaFX para aplicaciones de escritorio en Java, son ejemplos representativos de cómo diferentes plataformas ofrecen recursos que mejoran la productividad y calidad del diseño de la interfaz.

Al considerar las herramientas utilizadas para la edificación de interfaces, se puede notar la existencia tanto de soluciones propietarias, como Adobe XD o Microsoft Visual Studio, como de herramientas de código abierto, tales como Figma o GIMP. Las herramientas propietarias suelen ofrecer un conjunto de características robustas, con actualizaciones regulares y soporte técnico, lo que puede ser ventajoso para empresas que buscan estabilidad y funcionalidad. Por otro lado, las herramientas libres permiten una mayor flexibilidad y personalización, lo que habilita a los desarrolladores a adaptar la herramienta a sus necesidades específicas, fomentando también la colaboración y el acceso a recursos de la comunidad.

Los entornos de desarrollo integrados, comúnmente conocidos como IDEs, desempeñan un papel crucial en la creación y diseño de interfaces de usuario. Estas plataformas no solo proporcionan un editor de código, sino que también integran herramientas que optimizan el flujo de trabajo del desarrollador. IDEs como IntelliJ IDEA para Java, Visual Studio para .NET, o Android Studio para aplicaciones móviles ofrecen funcionalidades que permiten un diseño más intuitivo y amplio de las interfaces, mejorando así la eficiencia en el proceso de desarrollo. A través de interfaces gráficas, los IDEs permiten a los desarrolladores arrastrar y soltar componentes en las áreas de diseño, lo que facilita considerablemente la creación del layout deseado.

El área de diseño en la creación de interfaces es crítica y se refiere a la zona donde se construye gráficamente la interfaz de usuario. Esta área permite a los desarrolladores visualizar cómo se verá la aplicación final antes de implementarla, facilitando ajustes en tiempo real. Dentro de este entorno, el uso de contenedores es fundamental. Los contenedores son elementos que permiten agrupar y organizar otros componentes visuales en la interfaz, garantizando que presenten un diseño estructurado. Existen varios tipos de contenedores, como paneles, marcos y diseños de cuadrícula, cada uno con particularidades que influencian la disposición y el comportamiento de los elementos dentro de la interfaz. La elección del contenedor adecuado tiene un impacto significativo en la usabilidad y la estética de la aplicación.

La adición y eliminación de componentes dentro de la interfaz deben realizarse de manera controlada y consciente. Es necesario garantizar que la disposición mantenga una lógica accesible para el usuario final. La ubicación y el alineamiento de los componentes también son factores clave que determinan cómo interactúa el usuario con la aplicación. La apariencia y la funcionalidad de los elementos dependen de la forma en que se presentan en la interfaz, por lo que una correcta organización garantiza una experiencia más intuitiva.

Para facilitar la organización de componentes, se utilizan distintos tipos de administradores de diseño que controlan la disposición automática de los elementos en los contenedores. Los administradores de diseño son responsables de gestionar cómo se distribuyen los componentes a medida que la ventana de la aplicación cambia de tamaño. Cada tipo de contenedor puede tener administradores de diseño específicos asociados, que indicarán cómo reaccionan los elementos internos ante cambios en el tamaño o la orientación de la ventana.

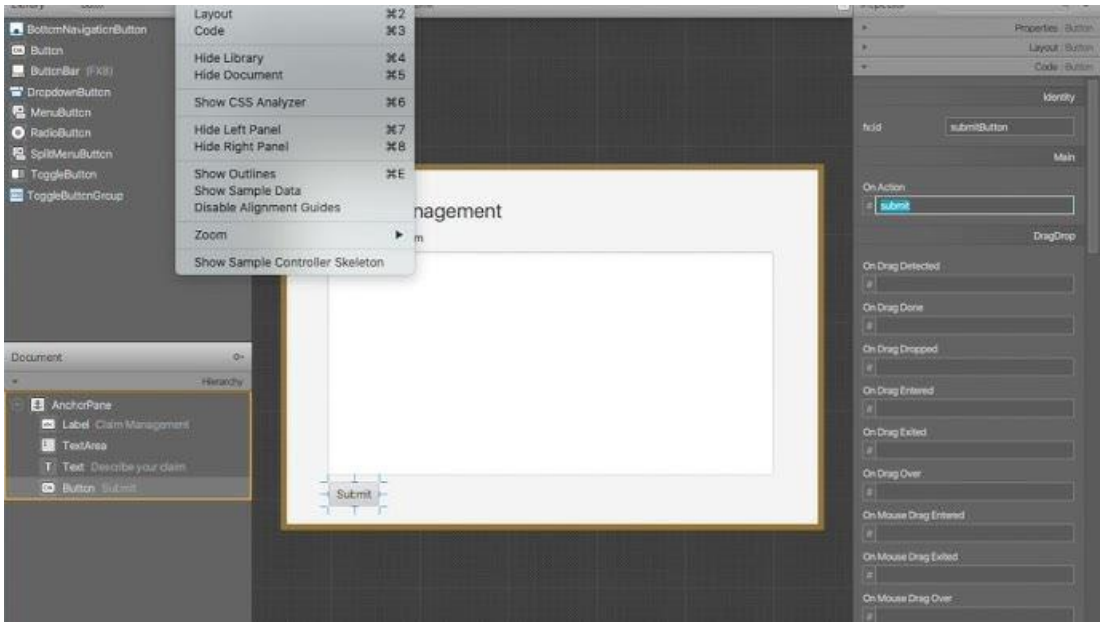


Ilustración 1. Área de trabajo de Scene Builder

Modificar propiedades dentro de la interfaz permite personalizar la experiencia del usuario, adaptando los elementos visuales a las necesidades de la aplicación. Esta modificación puede incluir cambios en el tamaño, color, tipo de letra y otras propiedades visuales. Igualmente, la asociación de acciones a eventos es esencial para hacer que los componentes respondan

adecuadamente a la interacción del usuario. Esto significa que se deben programar reacciones específicas que se ejecuten cuando se producen determinados eventos, como un clic del mouse, el paso del mouse sobre un componente o la entrada de datos en un campo de texto.

Los diálogos se presentan como una herramienta complementaria en la interfaz, distinguiéndose entre diálogos modales y no modales. Los diálogos modales requieren que el usuario interactúe con ellos antes de poder volver a la aplicación principal, lo que puede ser útil para captar la atención del usuario en situaciones críticas. Por el contrario, los diálogos no modales permiten que el usuario interactúe con la aplicación mientras el diálogo permanece abierto, ofreciendo más flexibilidad.

La edición del código generado por las herramientas de diseño es una capacidad significativa que permite a los desarrolladores realizar ajustes y personalizaciones que no siempre están disponibles a través de la interfaz visual. Este acceso al código subyacente da la posibilidad de optimizar el rendimiento o añadir características adicionales que van más allá de las capacidades de la herramienta de diseño.

Finalmente, en la creación de interfaces de usuario, es indispensable comprender el funcionamiento de clases, propiedades y métodos, así como la gestión de eventos asociados a los componentes. Las clases encapsulan la lógica y los comportamientos de un componente, mientras que las propiedades determinan sus características visuales y funcionales. Por otro lado, los métodos pueden ser invocados para ejecutar acciones específicas, mientras que la administración de eventos permite responder a las interacciones del usuario. La asimilación de estos conceptos resulta fundamental para el desarrollo eficaz de interfaces que sean funcionales, accesibles y que ofrezcan una experiencia satisfactoria al usuario.

1. LIBRERÍAS DE COMPONENTES PARA SISTEMAS OPERATIVOS Y LENGUAJES DE PROGRAMACIÓN

Existen diversas librerías de componentes que facilitan la creación de interfaces de usuario en diferentes sistemas operativos y lenguajes de programación. Estas librerías permiten a los desarrolladores construir aplicaciones de manera más eficiente, ya que proporcionan una amplia gama de componentes predefinidos, como botones, cuadros de texto, listas y menús, que pueden ser fácilmente integrados en las aplicaciones.

En el entorno de desarrollo de aplicaciones web, se destacan librerías como React, Angular y Vue.js. React, desarrollado por Facebook, permite la creación de interfaces interactivas mediante la construcción de componentes reutilizables. Angular, mantenido por Google, se utiliza para desarrollar aplicaciones de una sola página, incorporando un enfoque estructurado y una gestión eficiente del contenido dinámico. Vue.js es una librería progresiva que se integra fácilmente con otros proyectos y se utiliza para construir interfaces de usuario interactivas.

Para aplicaciones móviles, se encuentran librerías como Flutter y React Native. Flutter, de Google, permite crear aplicaciones nativas a partir de un solo código que se ejecuta en iOS y Android. React Native, por su parte, ofrece la posibilidad de desarrollar aplicaciones móviles utilizando JavaScript y React, lo que facilita la creación de interfaces que son consistentes con las plataformas móviles nativas.

En el desarrollo de aplicaciones de escritorio, librerías como Electron y Qt son comunes. Electron permite construir aplicaciones de escritorio utilizando tecnologías web como HTML, CSS y JavaScript, lo que permite un desarrollo multiplataforma. Qt es un marco de desarrollo muy utilizado para crear aplicaciones de escritorio, que proporciona herramientas para construir interfaces de usuario altamente personalizables y de alto rendimiento.

Otras librerías notables incluyen Xamarin para aplicaciones móviles, que permite a los desarrolladores utilizar C# y .NET para crear aplicaciones nativas en diferentes plataformas, y WPF (Windows Presentation Foundation) para aplicaciones de escritorio en Windows, que posibilita crear interfaces ricas utilizando XAML y C#.

Al seleccionar una librería de componentes para los proyectos, los desarrolladores deben considerar la compatibilidad, la comunidad de soporte y la documentación disponible, dado que estas características pueden influir en el proceso de desarrollo y en la experiencia del usuario final.

1.1. 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 selecciona 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.

Ilustración 2. Controles de Java Swing en la paleta de NetBeans

1.2. BIBLIOTECAS DE COMPONENTES

Los componentes que pueden formar parte de una interfaz gráfica se suelen presentar agrupados en **bibliotecas** (del inglés **library**) con la posibilidad de que el usuario pueda generar sus propios componentes y añadirlos o crear sus propias bibliotecas.

Las 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 JFC 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 AWT, 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** MSDN **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 (que se verán en la siguiente unidad temática). Normalmente disponen de mecanismos para elaborar las interfaces y traducirlas a diferentes lenguajes de programación, para después ser integradas en la aplicación final.

2. HERRAMIENTAS PROPIETARIAS Y LIBRES DE EDICIÓN DE INTERFACES

Las herramientas de edición de interfaces se dividen en herramientas propietarias y libres, cada una desempeñando un rol importante en el desarrollo de aplicaciones y la creación de experiencias de usuario. A continuación, se describen en detalle ambos tipos de herramientas, incluyendo ejemplos y casos de uso representativos.

2.1. HERRAMIENTAS PROPIETARIAS

Las herramientas propietarias requieren la compra de licencias y, en muchos casos, ofrecen un ecosistema más cerrado con soporte técnico y actualizaciones garantizadas. Estas soluciones suelen tener características avanzadas y un enfoque en la integración con otras aplicaciones de la misma marca.

Adobe XD

Adobe XD es una herramienta diseñada para el diseño y prototipado de aplicaciones. Su interfaz permite crear diseños vectoriales, así como integrar animaciones y transiciones de pantalla. Por ejemplo, en el desarrollo de una aplicación de comercio electrónico, un diseñador puede utilizar Adobe XD para crear un flujo de usuario que simule cómo el cliente pasaría de la pantalla de inicio al proceso de pago. Esto ayuda al equipo de desarrollo a comprender las interacciones necesarias y a recibir retroalimentación antes de que la implementación técnica comience.

Sketch

Esta herramienta tiene una gran aceptación en el desarrollo para plataformas Apple. Su sistema de símbolos permite la creación de componentes reutilizables, lo que es útil en proyectos que requieren consistencia. En un caso práctico, un grupo de diseñadores podría trabajar en un portal web para una institución educativa, donde crearían símbolos para botones de acción, menús desplegables y otros elementos de navegación. Al actualizar un símbolo, los cambios se reflejarán automáticamente en todos los lugares donde aparece, lo que ahorra tiempo y esfuerzo en el mantenimiento del diseño.

Figma

Aunque Figma se clasifica comúnmente como herramienta libre, también ofrece un modelo de suscripción que proporciona características adicionales. Es conocida por su naturaleza colaborativa en tiempo real, permitiendo que múltiples usuarios trabajen en el mismo archivo simultáneamente. En un proyecto desarrollo de una aplicación de fitness, los diseñadores y desarrolladores podrían utilizar Figma para crear y refinar la interfaz mientras efectúan ajustes en tiempo real, facilitando el proceso de feedback y toma de decisiones.

2.2. HERRAMIENTAS LIBRES

Las herramientas libres son de código abierto o tienen versiones gratuitas, y aunque pueden carecer de algunas características avanzadas de las soluciones de pago, son muy efectivas para la creación de interfaces debido a su flexibilidad y accesibilidad.

Figma (versión gratuita)

Aunque Figma ofrece una versión de pago, su versión gratuita es ampliamente utilizada y permite a los usuarios crear prototipos y colaborar. En un caso práctico, un grupo en un entorno educativo podría emplear Figma para diseñar una aplicación destinada a facilitar el aprendizaje en línea. Los diseñadores de gráficos podrían utilizar la versión gratuita para crear su interfaz y recibir retroalimentación de sus compañeros y profesores a través de comentarios en el diseño.

InVision

InVision se centra en el prototipado y permite a los diseñadores crear flujos interactivos. Cuando se trabaja en un proyecto para una aplicación de redes sociales, por ejemplo, los diseñadores pueden usar InVision para crear un prototipo que simule las interacciones de los usuarios al realizar tareas como crear publicaciones, emitir comentarios o compartir contenido. Esto permite al equipo presentar la interfaz a los inversores o directores de producto e identificar áreas donde el flujo de la aplicación puede mejorarse.

Gravit Designer

Esta herramienta es libre y ofrece funcionalidades similares a las herramientas de pago en cuanto a diseño vectorial. En el desarrollo de interfaz para una aplicación de productividad, un diseñador podría utilizar Gravit Designer para crear íconos personalizados y elementos gráficos que se integrarán en la aplicación. Su versatilidad permite exportar diseños en varios formatos, facilitando la colaboración con desarrolladores que requieren recursos en diferentes tipos de archivo.

Penpot

Penpot es otra herramienta de diseño de interfaces de código abierto que permite la creación de prototipos y el diseño colaborativo. En un proyecto de software para una ONG, los diseñadores podrían utilizar Penpot para discutir y ajustar la interfaz de una aplicación que ayuda a los voluntarios a coordinar esfuerzos de ayuda. La naturaleza de código abierto de Penpot permite que los diseñadores personalicen la herramienta según sus necesidades específicas.

2.3. PRIMEROS PASOS EN LA ELECCIÓN DE HERRAMIENTAS

La selección de una herramienta de edición de interfaces debe considerar varios factores, incluidos el presupuesto, los requerimientos de proyecto y la familiaridad del equipo con la herramienta. En proyectos grandes, donde la escala y la colaboración son importantes, herramientas como Figma o Adobe XD pueden ser más adecuadas debido a sus características

de trabajo en grupo. Para proyectos pequeños o iniciativas personales, las herramientas libres pueden ser suficientes y ofrecen flexibilidad sin costo.

Además, la integración con otros entornos de trabajo es otro aspecto a considerar. Las herramientas que se integran fácilmente con sistemas de gestión de proyectos, como Jira o Trello, pueden facilitar la colaboración entre los equipos de desarrollo y diseño.

Otro aspecto son los recursos de aprendizaje y la comunidad que rodea a cada herramienta. Las herramientas libres frecuentemente tienen comunidades robustas que comparten conocimientos, proporcionando una base de soporte que puede ser muy útil. Por ejemplo, Figma cuenta con tutoriales en línea y foros donde los usuarios pueden compartir sus experiencias y resolver problemas comunes.

Ilustración 3. Logo de Figma

La elección entre herramientas propietarias y libres se fundamenta en aspectos prácticos, como las necesidades del proyecto, y la cultura del equipo de trabajo, donde el diseño y desarrollo colaborativo se pueden llevar a cabo de manera efectiva y eficiente. Las herramientas de diseño de interfaces seguirán evolucionando, adaptándose a las nuevas tecnologías y a los requisitos cambiantes del mercado de software, ofreciendo a los equipos de desarrollo y diseño procesos cada vez más integrados y eficientes.

3. IDE QUE NOS PERMITEN DESARROLLAR INTERFACES

Existen diferentes Entornos de Desarrollo Integrados (IDE) que permiten la creación de interfaces de usuario de manera eficiente. Cada IDE presenta características específicas que se adaptan a variadas necesidades de desarrollo. Algunos de estos entornos ofrecen herramientas de diseño visual que posibilitan construir interfaces mediante técnicas de arrastrar y soltar elementos, lo que reduce la necesidad de escribir código y facilita la visualización del diseño final.

Los IDE más utilizados en la actualidad son los siguientes:

- *Eclipse*
- *NetBeans*
- *Microsoft Visual Studio*
- *Xcode*
- *JDeveloper*

La elección del IDE adecuado para el desarrollo de interfaces de usuario depende de diversos factores como el tipo de aplicación, el lenguaje de programación y las preferencias personales del programador.

3.1. ECLIPSE

Eclipse es un entorno de desarrollo integrado (IDE) de código abierto que se destaca en la creación de aplicaciones en Java, aunque también es compatible con otros lenguajes de programación. Su arquitectura basada en plugins permite a los desarrolladores integrar diversas herramientas y bibliotecas, como Swing y JavaFX, lo que lo convierte en una herramienta versátil para la realización de interfaces gráficas de usuario (GUI).

JavaFX es un framework que permite crear interfaces modernas y atractivas. Este también se integra eficientemente en Eclipse, ofreciendo controles ricos como tablas interactivas, gráficos y efectos visuales. Por ejemplo, en una aplicación de análisis de ventas, un desarrollador puede utilizar JavaFX para representar visualmente los datos de ventas mediante gráficos de barra, lo que permite a los usuarios interpretar rápidamente la información. JavaFX permite también incorporar animaciones, mejorando la experiencia del usuario al interactuar con los elementos de la interfaz.

3.2. NETBEANS

Desarrollar interfaces gráficas en NetBeans, utilizando Java, se centra en la facilidad de uso del entorno integrado de desarrollo (IDE) y las herramientas visuales que ofrece. NetBeans proporciona un editor de diseño gráfico, conocido como **Swing GUI Builder**, que permite a los desarrolladores diseñar interfaces arrastrando y soltando componentes visuales como botones, etiquetas, cuadros de texto y paneles. Este enfoque visual facilita la construcción de interfaces

complejas sin necesidad de escribir manualmente código extenso, ya que el IDE genera automáticamente el código necesario en segundo plano. Esto acelera significativamente el proceso de desarrollo y minimiza errores asociados a la manipulación directa de los componentes.

Otro aspecto esencial es la integración de NetBeans con el modelo de eventos de Java, que es fundamental para que las interfaces respondan a las acciones del usuario. En NetBeans, se pueden configurar fácilmente los **eventos** asociados a los componentes gráficos, como clics de botón o cambios en cuadros de texto, mediante la interfaz visual. El IDE permite asociar métodos específicos a estos eventos sin necesidad de salir del entorno gráfico, lo que facilita la creación de respuestas interactivas en la aplicación. Esta integración entre diseño visual y lógica de programación es uno de los puntos fuertes del uso de NetBeans para desarrollar interfaces gráficas.

3.3. VISUAL STUDIO

Visual Studio es un entorno de desarrollo integrado (IDE) que proporciona una amplia gama de funcionalidades para la creación y gestión de aplicaciones. Este IDE se utiliza para comprender a fondo las herramientas y su aplicación en la elaboración de interfaces de usuario efectivas.

El editor visual de Visual Studio permite trabajar en la interfaz de usuario mediante un sistema de arrastrar y soltar. Esta herramienta es útil en aplicaciones de Windows Forms y WPF (Windows Presentation Foundation), que son marcos comunes para desarrollar aplicaciones de escritorio en Windows. En Windows Forms, un desarrollador puede crear una aplicación básica de calculadora arrastrando botones para los dígitos y las operaciones sobre el espacio de trabajo. Al configurar eventos como los clics de botones, es posible escribir el código en el lenguaje de programación elegido para llevar a cabo las operaciones necesarias.

Con WPF, el diseño de la interfaz se realiza utilizando XAML (Extensible Application Markup Language), lo que permite la definición de la interfaz de manera declarativa. Este enfoque no solo optimiza el despliegue visual, sino que también proporciona una separación clara entre la lógica de la aplicación y la interfaz de usuario.

3.4. XCODE

Xcode es un entorno de desarrollo integrado (IDE) de Apple, utilizado ampliamente para crear aplicaciones en diversas plataformas como iOS, macOS, watchOS y tvOS. Este IDE permite a los desarrolladores diseñar, codificar y probar aplicaciones en una única plataforma, lo que facilita el flujo de trabajo y la colaboración en proyectos de programación.

Una característica destacable de Xcode es su integración con Interface Builder. Esta herramienta gráfica permite crear interfaces de usuario de manera visual, facilitando la disposición de elementos como botones, etiquetas y controles. Al utilizar Interface Builder, los desarrolladores pueden arrastrar y soltar componentes, ajustando sus propiedades directamente desde una vista gráfica. Por ejemplo, en la construcción de una aplicación de registro de usuarios, un desarrollador puede arrastrar un campo de texto para el correo electrónico y un botón de

"Enviar", sin necesidad de escribir manualmente el código de diseño. Esto acelera el proceso de creación y permite ver el resultado en tiempo real.

3.5. JDEVELOPER

Desarrollar interfaces gráficas con JDeveloper, especialmente dentro del entorno de Oracle, se enfoca en el uso de la tecnología **Oracle Application Development Framework (ADF)**. ADF es un marco de trabajo que facilita el desarrollo de aplicaciones empresariales complejas, proporcionando herramientas que permiten crear interfaces de usuario avanzadas con menos esfuerzo de codificación manual. JDeveloper ofrece un diseñador visual similar a un editor WYSIWYG (What You See Is What You Get), donde los desarrolladores pueden arrastrar y soltar componentes visuales, como formularios, tablas y gráficos, directamente sobre la página, mientras el IDE genera automáticamente el código necesario para estos elementos.

Un aspecto clave del desarrollo con JDeveloper es su enfoque en el **desarrollo basado en modelos** (MDA, por sus siglas en inglés). A través de ADF, los desarrolladores pueden trabajar con una arquitectura basada en el patrón MVC (Modelo-Vista-Controlador), donde la lógica de negocio, la interacción con bases de datos y la interfaz de usuario están claramente separadas. Esto no solo mejora la organización del código, sino que también facilita el mantenimiento y la escalabilidad de la aplicación.

4. ÁREA DE DISEÑO

El área de diseño se refiere al espacio dentro de un entorno de desarrollo donde se crean y organizan visualmente los elementos que conforman una aplicación. Este entorno permite la disposición gráfica de los componentes, facilitando la creación de prototipos y la visualización de cómo se verá la aplicación en diferentes dispositivos. En este espacio, es posible manipular diversos aspectos de los elementos, como su tamaño, color, posición y comportamiento, para asegurar que se adapten a los requerimientos del proyecto y a las expectativas del usuario.

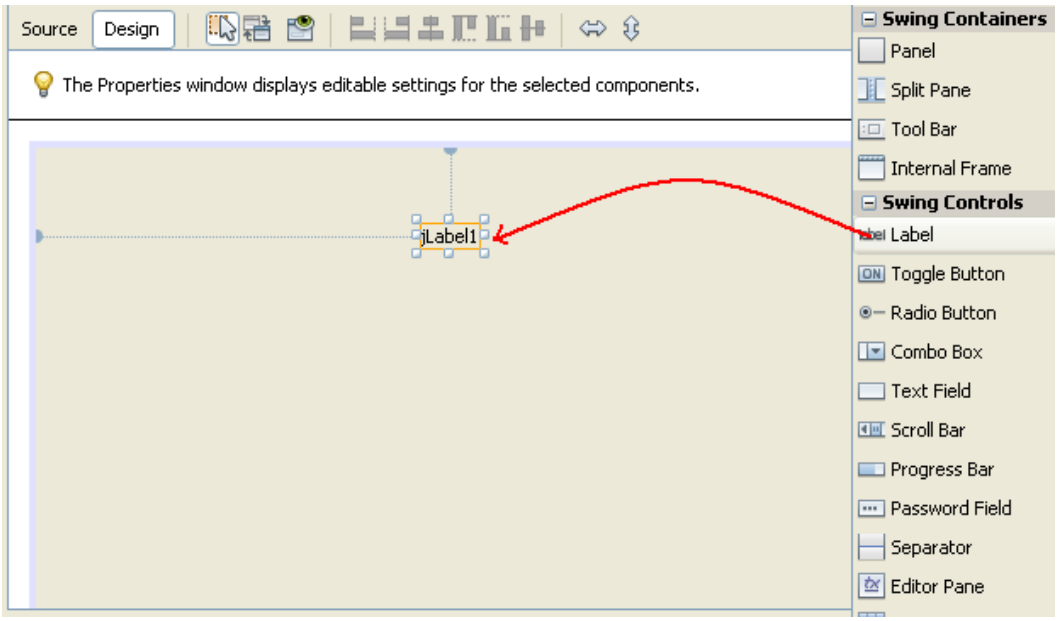


Ilustración 4. Área de diseño de NetBeans

Una característica destacada del área de diseño es la capacidad de arrastrar y soltar componentes desde una biblioteca, lo que permite construir la interfaz de manera más intuitiva y rápida. Esto ahorra tiempo y promueve la experimentación con diferentes diseños y estilos. Además, se incorpora la visualización en tiempo real, lo que significa que los cambios realizados en los elementos se reflejan de inmediato en la interfaz, facilitando la iteración y ajuste de los diseños.

El área de diseño también incluye guías de alineación y patrones que ayudan a mantener la coherencia estética y funcional en la aplicación. Esto contribuye a crear experiencias de usuario óptimas, dado que una buena alineación y distribución de elementos favorecen una navegación más sencilla y un uso más eficiente de la interfaz. Algunos entornos de desarrollo ofrecen herramientas de validación que permiten comprobar si los componentes cumplen con los estándares de accesibilidad y usabilidad.

5. PALETA DE COMPONENTES

La paleta de componentes es una colección sistemática de elementos de interfaz que permite a los diseñadores y desarrolladores construir aplicaciones de manera eficiente y coherente. Esta colección incluye una variedad de elementos que, al ser utilizados apropiadamente, facilitan la interacción del usuario con el sistema. A continuación, se detallan varios tipos de componentes comunes en el desarrollo de interfaces de usuario.

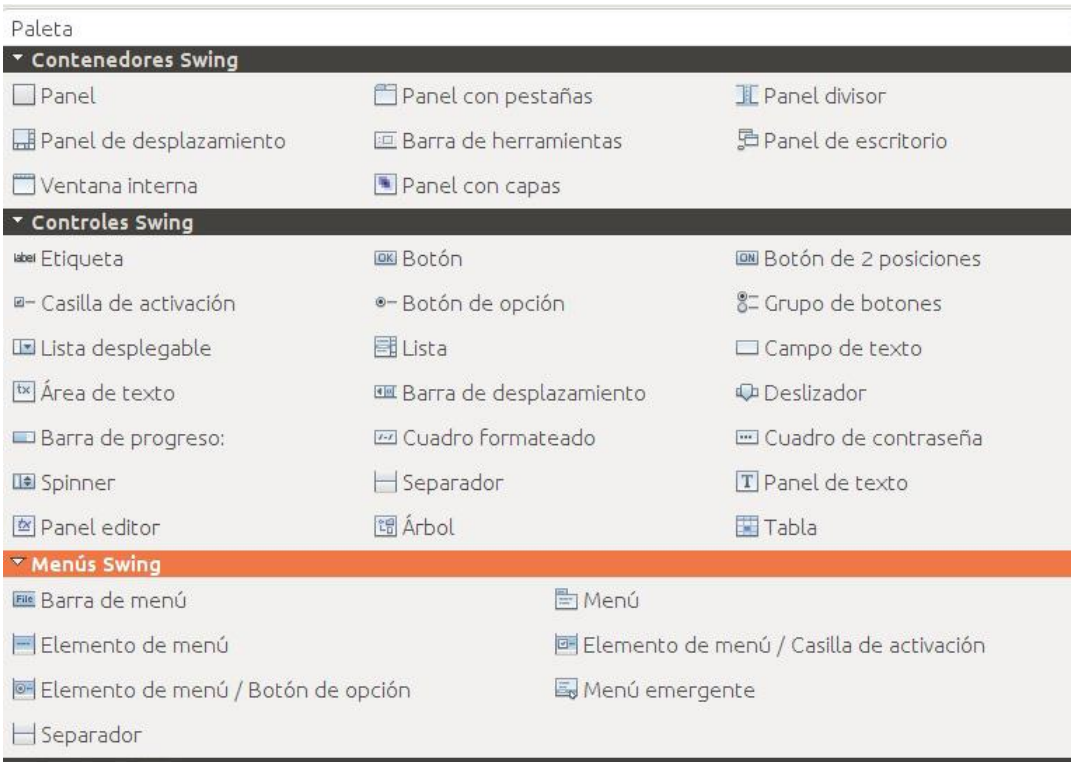


Ilustración 5. Paleta de componentes de NetBeans

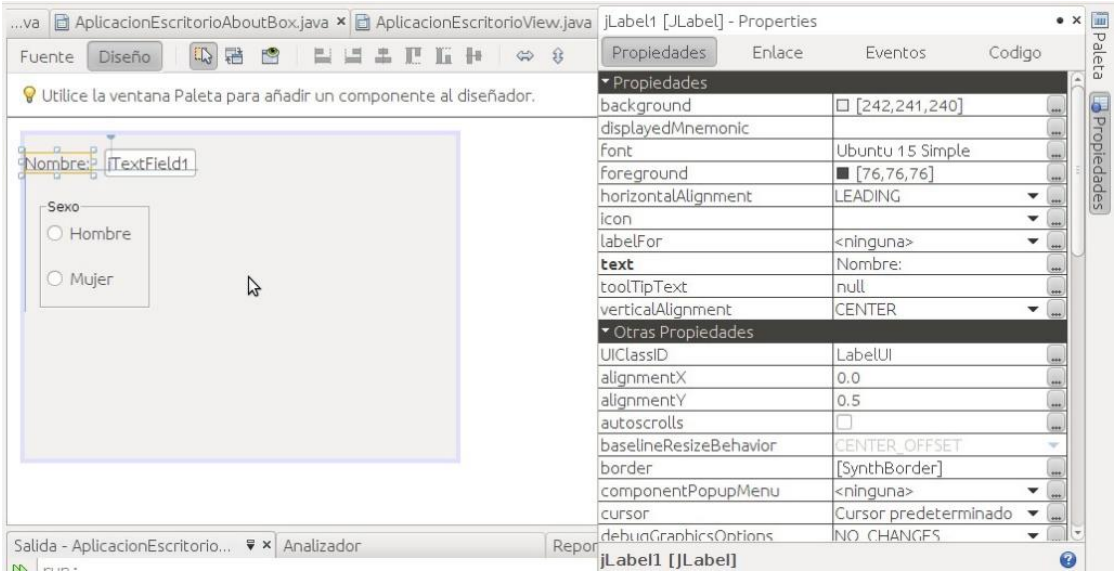
6. EDITOR DE PROPIEDADES

El editor de propiedades es una herramienta importante en el proceso de modificación de atributos de los elementos de la interfaz de usuario, así como en el establecimiento de estilos visuales y comportamientos interactivos. Su utilización es relevante en el diseño de aplicaciones, ya que brinda una forma eficiente de personalizar y optimizar la experiencia del usuario.

6.1. PROPIEDADES VISUALES

Las propiedades visuales son características que determinan la presentación de un componente en la interfaz. Esto incluye aspectos como el color, los márgenes, el tamaño y el estilo de fuente. Al utilizar un editor de propiedades, el desarrollador puede ajustar cada uno de estos atributos de manera precisa.

Por ejemplo, al diseñar una aplicación de gestión de proyectos, se puede crear un panel que contenga botones para acciones como "Agregar tarea", "Eliminar tarea" y "Guardar cambios". A través del editor de propiedades, el desarrollador puede configurar el color de fondo de los botones para que estén alineados con la paleta de la aplicación. Además, es posible ajustar el tamaño y tipo de fuente para asegurar que sean legibles. La modificación de márgenes también puede contribuir a mejorar la estética y la usabilidad.



6.2. PROPIEDADES FUNCIONALES

Más allá de las propiedades visuales, el editor de propiedades permite establecer atributos que determinan cómo los componentes interactúan con el usuario. Esto incluye la vinculación de eventos como clics, desplazamientos y entradas de texto.

Por ejemplo, en una interfaz para un sistema de reservas de vuelos, el botón "Buscar vuelos" puede tener un evento de clic vinculado que ejecuta una función para consultar una base de datos de vuelos. Al seleccionar el botón en el editor de propiedades, el desarrollador puede asociar el evento correspondiente, definiendo la acción que debe llevarse a cabo tras la

interacción. Además, se puede ajustar la propiedad de habilitación del botón para que esté activa solo cuando el usuario complete los campos requeridos, como la selección de origen y destino.

6.3. PERSONALIZACIÓN DE ESTADOS

El editor de propiedades permite también la personalización de los estados de los componentes. Cada elemento puede tener diferentes representaciones visuales y comportamientos dependiendo de su estado.

Por ejemplo, un botón en estado normal puede ser de color verde, pero cuando el cursor se coloca sobre él, podría cambiar a un color más oscuro o brillante, indicando que es interactivo. Este efecto se puede lograr mediante la definición de propiedades que manejen eventos como "mouseover" y "mouseout". Esta técnica mejora la interactividad de la interfaz frente a las acciones del usuario. Un caso práctico podría ser un formulario de contacto donde cada campo tiene un borde que cambia de color al interactuar, proporcionando una confirmación visual de que el usuario se encuentra en el campo adecuado.

7. CONTENEDORES Y COMPONENTES DE LA INTERFAZ

Para el desarrollo de los siguientes apartados nos centraremos en el lenguaje de programación Java y en el desarrollo de interfaces con NetBeans:

7.1. CONTENEDORES

Los contenedores son elementos importantes en el desarrollo de interfaces de usuario, ya que organizan y estructuran los componentes visuales dentro de una aplicación. Actúan como agrupaciones que facilitan el manejo de múltiples elementos, permitiendo que estos se comporten como una unidad, lo que optimiza la gestión y la presentación de datos.

Al utilizar contenedores, se mejora la experiencia del usuario al asegurar que los elementos se alineen correctamente y mantengan una jerarquía de información clara. Esto resulta relevante en entornos multiplataforma, donde las diferencias en tamaño de pantalla y resolución requieren adaptabilidad y coherencia en el diseño.

Un formulario es una ventana que cuenta con tres botones: minimizar, maximizar y cerrar. También incluye una barra de título y está rodeado por bordes. Constituye la base para desarrollar una aplicación de escritorio. Sobre el formulario se pueden agregar controles o componentes, ya sean sencillos como botones y cuadros de texto, o más avanzados, como menús y rejillas de datos, que le otorgan funcionalidad:

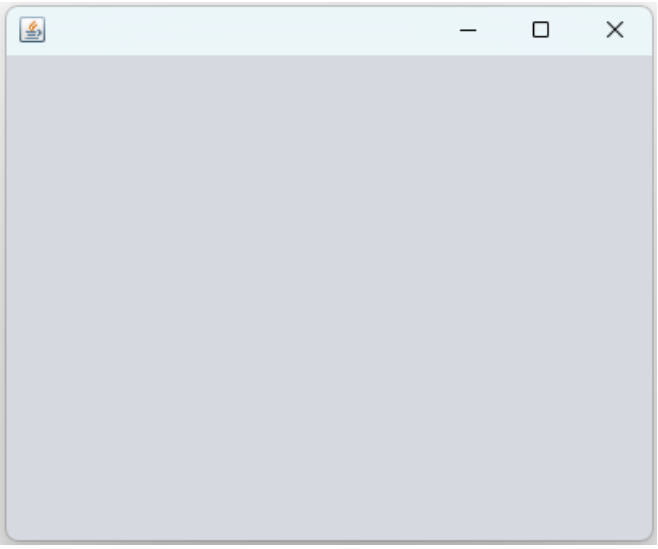


Ilustración 6. Ventana creada con un contenedor de nivel superior

Los contenedores pueden tener propiedades que controlan su comportamiento, como el posicionamiento, el ajuste de tamaño y el flujo de contenido. Estos aspectos son importantes para asegurar que la interfaz se muestre de forma adecuada en diferentes dispositivos, manteniendo la usabilidad y la estética del producto final.

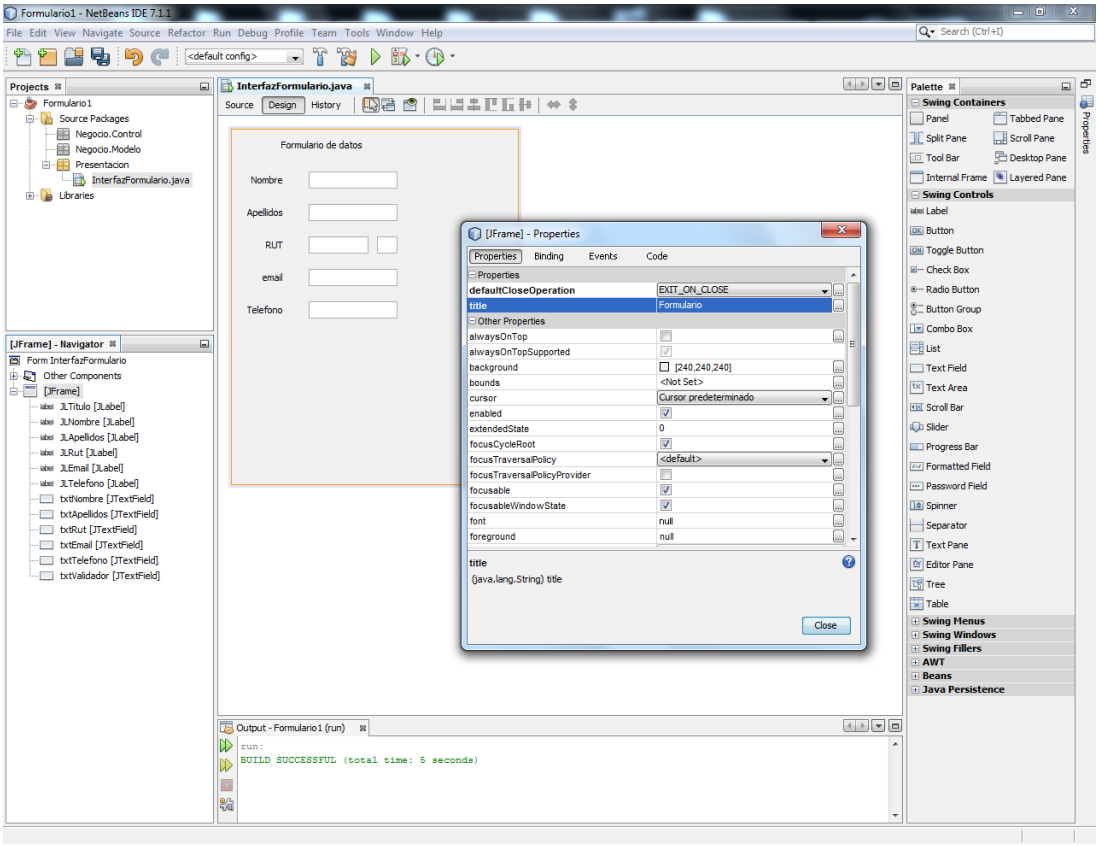


Ilustración 7. Ejemplo de diseño de formulario en el entorno de NetBeans

7.1.1. Contenedores de nivel superior

Una aplicación de escritorio en NetBeans se compone de varios formularios. Para crear un formulario, se debe utilizar un contenedor Java, un componente que permite incluir otros elementos, incluidos otros contenedores, que se emplearán para organizar los controles. Debido a esta capacidad, los contenedores forman una estructura jerárquica.

Un formulario se basa en un contenedor especial conocido como contenedor de nivel superior. Este tipo de contenedor incluye un panel de contenido (contentpane), que permite agregar otros componentes, incluidos contenedores adicionales que facilitan la distribución de los elementos.

Como contenedor de nivel superior en un formulario, se puede elegir entre una ventana (JFrame), un diálogo (JDialog) o un applet (JApplet), dependiendo de las necesidades. Todos estos componentes derivan de la clase Window en la jerarquía de clases de Java, que representa una ventana típica.

- **Ventana (JFrame):** es un formulario con título, bordes y los botones para maximizar, minimizar o cerrar. Por defecto, incluye un ícono en forma de taza de café, el cual se puede modificar, y también puede contener una barra de menú.
- **Diálogo (JDialog):** son formularios que suelen utilizarse para solicitar información al usuario. Su característica principal es que pueden ser modales o no. Un diálogo modal captura todas las entradas de usuario, impidiendo que se active otra ventana.
- **Applet (JApplet):** es una ventana que ejecuta una aplicación Java dentro del contexto de una página web.

En una aplicación de escritorio, generalmente se crea una ventana principal de tipo JFrame, y si se requieren ventanas secundarias, se utilizan otras ventanas o diálogos. 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.

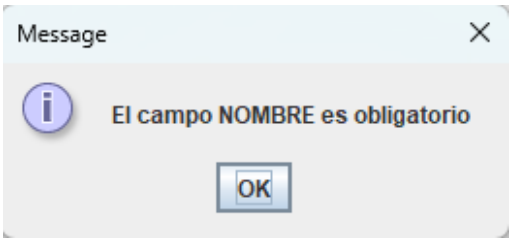


Ilustración 8. JOptionPane

- **Selector de archivos (JFileChooser)**: permite seleccionar un archivo del sistema de archivos del equipo donde se ejecuta la aplicación.

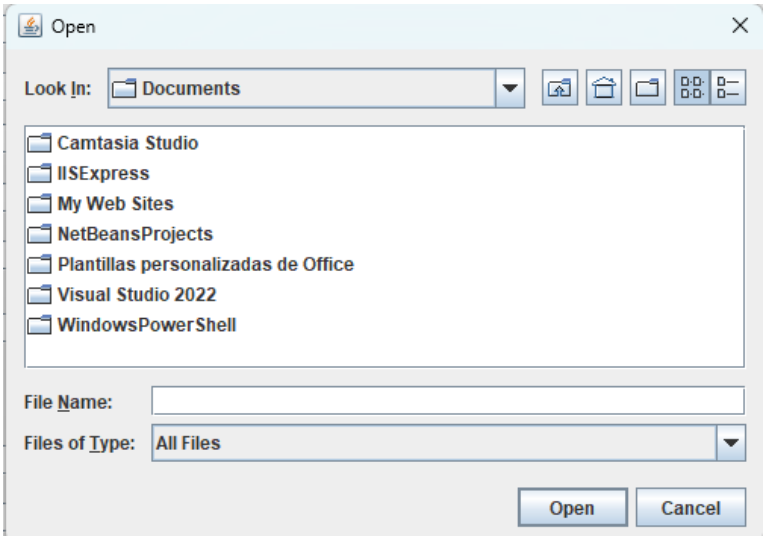


Ilustración 9. JFileChooser

- **Selector de colores (JColorChooser)**: permite seleccionar entre un conjunto de colores y devolverlo usando el código adecuado.

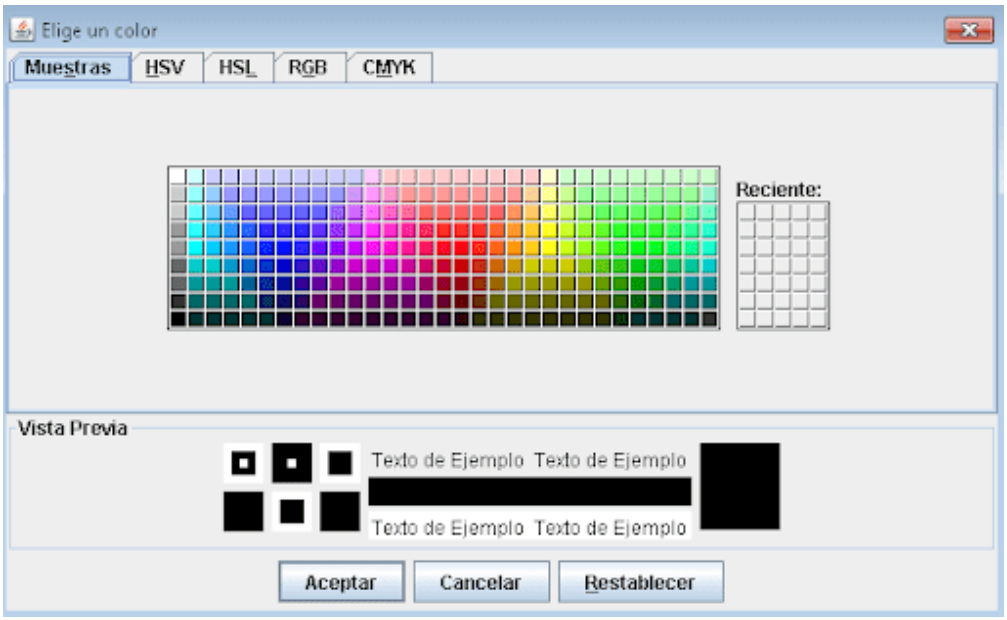


Ilustración 10. JColorChooser

7.1.2. Contenedores secundarios

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.

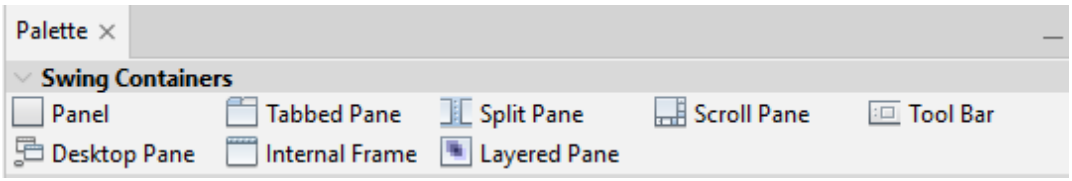


Ilustración 11. Contenedores de la librería Java Swing

7.2. COMPONENTES

Los componentes son elementos que conforman el diseño y desarrollo de interfaces de usuario en aplicaciones. Se consideran como bloques de construcción que permiten la interacción del usuario con el sistema. Existen diversos tipos y funciones de componentes que incluyen botones, campos de texto, menús, listas, diálogos y paneles, entre otros. Cada uno de estos elementos presenta una estructura y propiedades específicas que determinan su comportamiento y apariencia dentro de la interfaz.

Algunos componentes facilitan la entrada de datos, como los formularios, donde se puede ingresar información. Otros se dedican a mostrar datos, como etiquetas o tablas, organizando visualmente la información para facilitar su comprensión. Los componentes interactivos permiten la acción directa del usuario, como botones de navegación o controles deslizantes.

La disposición y organización de los componentes en la interfaz son aspectos relevantes en la experiencia del usuario. Una adecuada estructuración facilita la navegación y el uso de la aplicación, permitiendo a los usuarios encontrar rápidamente las funcionalidades que buscan. Los desarrolladores deben tener en cuenta factores como la jerarquía visual y la accesibilidad al diseñar la distribución de los componentes en la pantalla.

Un componente se identifica por su clase, que determina su apariencia y funcionalidad, y por su nombre, que lo distingue dentro de la aplicación. Al ser un objeto, un componente posee una serie de propiedades, según la clase a la que pertenezca, que pueden ser modificadas para adaptarlo, como el texto que muestra o su color.

La disposición de los componentes en el formulario sigue ciertas reglas conocidas como Layout, que dictan el orden y la posición en que deben aparecer. Estas disposiciones pueden organizarse en torno a los bordes del formulario (norte, sur, este y oeste), en forma de rejilla, o en un flujo continuo, ya sea en una o varias filas.

Cuando un componente puede interactuar con el usuario, su gestión se realiza a través de lo que se conoce como manejo de eventos. Un evento es una acción realizada sobre el componente que debe desencadenar una respuesta. La gestión de esta respuesta se lleva a cabo mediante los manejadores de eventos, que son funciones específicas asociadas a un elemento del componente llamado escuchador, donde se programa la acción a ejecutar.

Veamos todas estas cosas con un poco más de detenimiento.

7.2.1. 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, y mientras no se use queda replegada. En la patela de NetBeans los componentes están organizados en las siguientes categorías:

- **Contenedores**: son secundarios en la jerarquía de contenedores y se usan para distribuir y organizar el resto de controles.

- **Controles**: básicos para crear una interfaz útil para comunicarse con el usuario y mostrar o solicitar información.
- **Menús**: incluyen los controles necesarios para crear menús de aplicación complejos, con varios bloques, elementos activos e inactivos, etc. y menús contextuales (Popup Menu)
- **Ventanas**: 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.

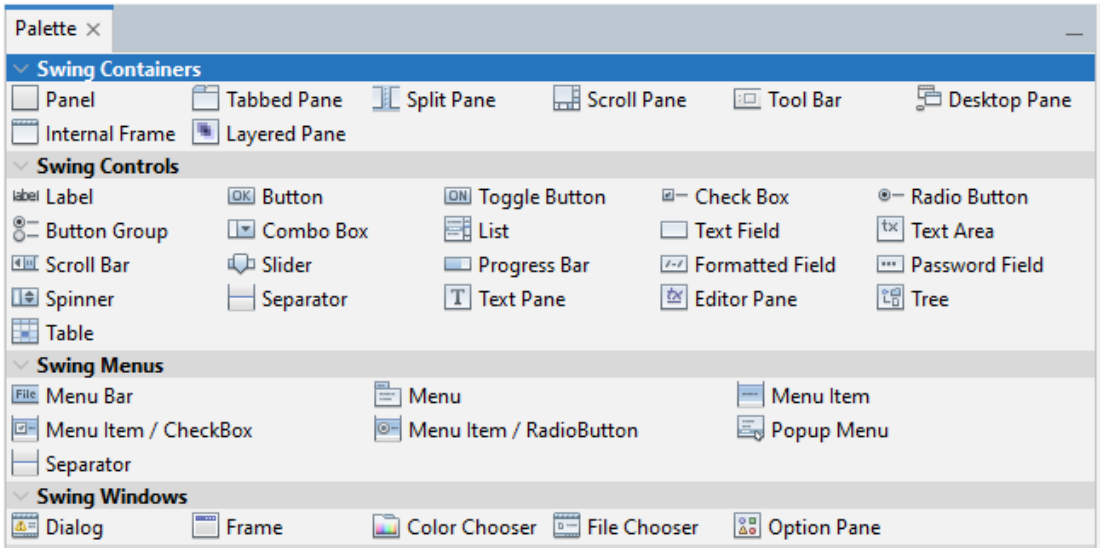


Ilustración 12. Componentes de la librería Java Swing

Para agregar componentes a la interfaz, selecciona el control en la paleta y haz clic en el área donde estás construyendo la interfaz. El control aparecerá con su aspecto predeterminado, que puedes modificar. Si arrastras el control, podrás moverlo dentro de su contenedor, y si haces clic en una esquina y desplazas el ratón, cambiarás su tamaño.

Al colocar un control en un formulario, se mostrarán guías que te ayudarán a ubicarlo con mayor precisión, siempre que tengas activa la opción de diseño libre. Puedes verificar esta opción en el inspector, haciendo clic con el botón derecho sobre el nodo raíz de la interfaz y seleccionando "activar gestor de distribución". Al mover un control, estas guías te permitirán alinearlo fácilmente con otros componentes. Para eliminar un control, simplemente lo seleccionamos y presionamos la tecla Supr, o selecciona la opción "Suprimir" en el menú contextual.

7.2.2. Modificación de propiedades

Una vez que has colocado un control en el formulario podemos modificar sus propiedades para adaptarlo a nuestras necesidades. Con el control seleccionado acceder 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 la variable...* En la imagen puedes ver las propiedades de una etiqueta que hemos añadido al formulario.

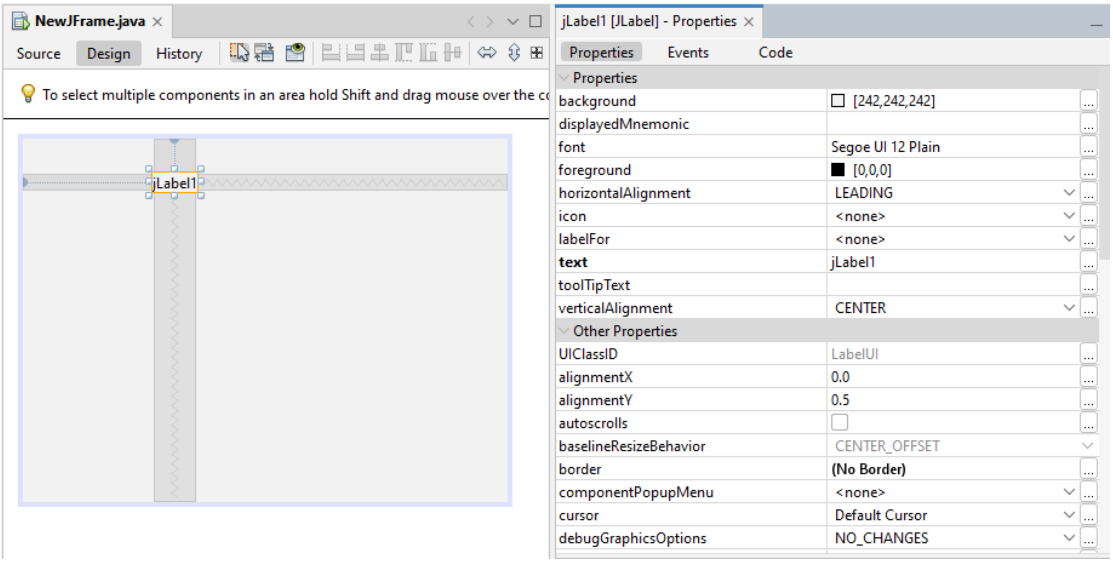


Ilustración 13. Propiedades del elemento etiqueta

7.2.3. Ubicación y alineamiento de los componentes

Internamente la herramienta emplea el mecanismo de Java para disponer los elementos llamado Layout, distribución o diseño. El Layout manager es el encargado de decidir en qué posiciones se mostrarán los componentes, que tamaño tendrán, que porción del contenedor abarcarán, etc. Swing dispone de ocho tipos de distribuciones:

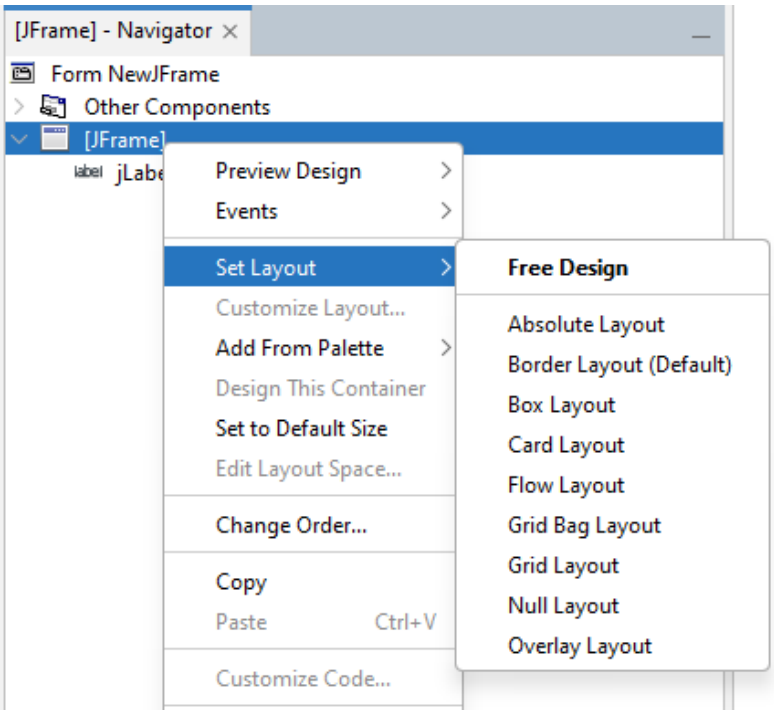


Ilustración 14. Tipos de Layout en Java Swing

Los administradores por defecto asociados a los contenedores son elementos importantes en la configuración y disposición de los componentes dentro de las interfaces de usuario. Cada tipo

de contenedor tiene un administrador específico que gestiona la colocación de los elementos que alberga, optimizando la organización y la accesibilidad.

La elección del administrador por defecto apropiado dependerá de las necesidades específicas del diseño y la funcionalidad deseada. Es importante evaluar los objetivos y la disposición de la interfaz antes de decidir qué administrador utilizar, considerando que en muchos casos se pueden combinar diferentes administradores en una misma interfaz para lograr un diseño más adaptado a requisitos particulares. Esta capacidad de personalización y combinación es uno de los aspectos más valiosos en el desarrollo de interfaces de usuario, ya que proporciona la flexibilidad necesaria para adaptarse a diferentes situaciones y preferencias.

A continuación, se analizan algunos de los administradores más utilizados y sus casos de uso.

Absolute Layout

Se colocan los componentes utilizando coordenadas exactas (x, y). A diferencia de otros gestores de distribución, como `FlowLayout` o `GridLayout`, donde los componentes se posicionan automáticamente según ciertas reglas, con `Absolute Layout` el programador tiene control total sobre la posición y tamaño de cada componente, siendo posible el solapamiento de los componentes en el contenedor:

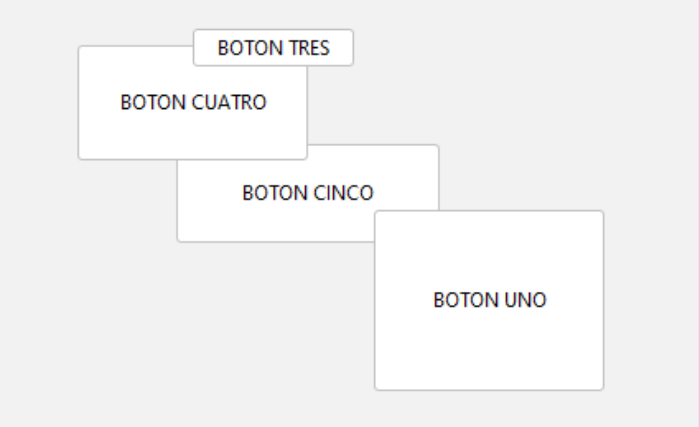


Ilustración 15. Distribución Absolute Layout

Border Layout

Este layout distribuye los componentes en cinco zonas predeterminadas: norte (NORTH), sur (SOUTH), este (EAST), oeste (WEST) y centro (CENTER). Si no especificamos ninguna región, por defecto el componente se inserta en el centro del contenedor.

El BorderLayout es el administrador por defecto que se utiliza en JFrame. Un caso práctico puede ser un panel de control para una aplicación de gestión de proyectos. En la parte norte se pueden incluir botones de navegación y un título. El área sur puede contener información adicional o botones de acción como "Guardar" y "Cancelar". En el lado este, se puede situar una lista de tareas pendientes y en el oeste un menú lateral de categorías, mientras que el centro se reserva para un área de visualización de detalles del proyecto. Esta organización permite un acceso fácil al contenido principal sin distracciones innecesarias.

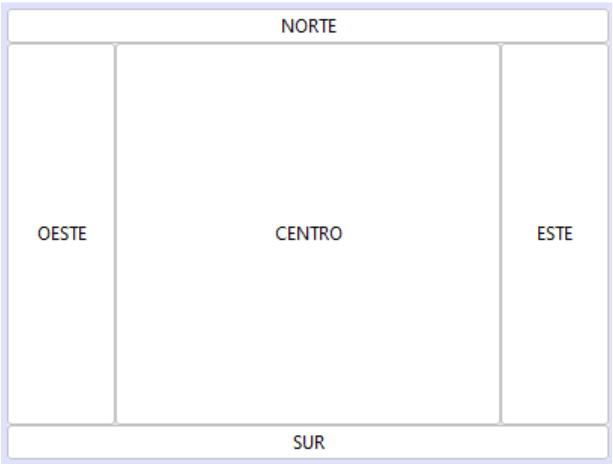


Ilustración 16. Distribución Border Layout

Box Layout

El Box Layout permite alinear componentes de manera vertical u horizontal, lo que proporciona flexibilidad al implementar formularios o listas de elementos. Este administrador facilita la disposición de los componentes uno al lado del otro, o uno encima del otro.

En la creación de un formulario de suscripción a un boletín informativo, se puede usar Box Layout en orientación vertical. Los campos de texto para el nombre y la dirección de correo electrónico se pueden colocar uno debajo del otro, acompañados de etiquetas descriptivas. Esta organización aprovecha el espacio vertical de forma eficiente, permitiendo a los usuarios completar el formulario de un modo ordenado.

Card Layout

organiza los componentes en un contenedor de manera que solo uno de ellos se muestra a la vez, lo que permite crear interfaces de usuario que simulan un efecto de "páginas". Este tipo de disposición es útil en asistentes de configuración o formularios con varias etapas.

Un ejemplo práctico sería un asistente de instalación de software. En cada paso del asistente, se puede emplear Card Layout para mostrar distintos paneles que guían al usuario a través del proceso de configuración, como aceptar términos de uso, seleccionar la carpeta de instalación y finalizar. Los botones de navegación permiten avanzar o retroceder entre etapas, haciendo que el proceso sea claro y ordenado.

Flow Layout

Es el administrador por defecto en JPanel en Java Swing, que organiza los componentes en una secuencia horizontal de izquierda a derecha. Si el espacio en el contenedor se agota, los componentes que no caben se ajustan automáticamente a una nueva línea. Este comportamiento es útil para crear barras de herramientas o grupos de botones.

Un ejemplo de aplicación puede ser una interfaz de usuario para una galería de fotos que muestre imágenes en miniatura. Al utilizar FlowLayout, las imágenes se disponen una al lado de

la otra, y al llegar al final del contenedor se pasan a la siguiente línea, manteniendo un diseño limpio y accesible, lo que permite al usuario navegar a través de las imágenes de manera fluida.

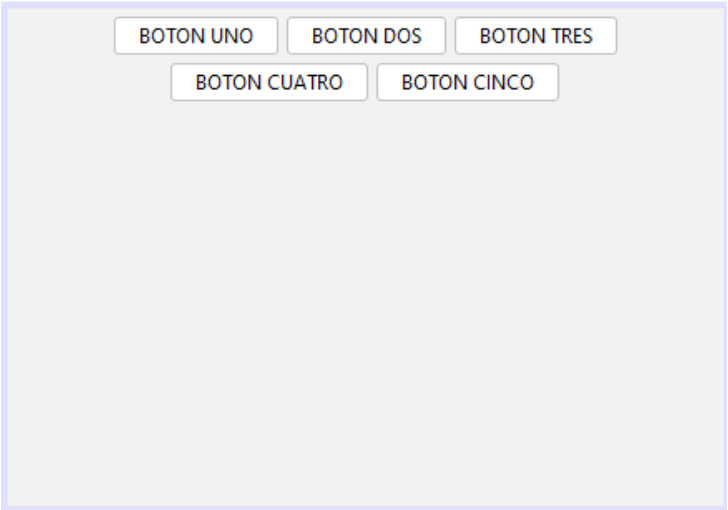


Ilustración 17. Distribución Flow Layout

Grid Layout

Organiza los componentes en una cuadrícula de filas y columnas uniformes. Este administrador asegura que los componentes tengan un tamaño igual, lo que ayuda a mantener una apariencia ordenada y homogénea.

Un ejemplo de uso se puede ver en una interfaz para una calculadora gráfica, donde se puede emplear Grid Layout para establecer la disposición de los botones numéricos y de operación. Un diseño de 4 filas y 4 columnas podría contener los números del 1 al 9 en las primeras tres filas y el 0 en la última fila, junto con botones de operación como suma y resta. Este enfoque resulta intuitivo, ya que los usuarios pueden acceder a los números y realizar operaciones con facilidad.

Ilustración 18. Distribución Grid Layout

GridBag Layout

Semejante a `Grid Layout`, pero permite a un componente que ocupe más de una celda. Es un administrador que proporciona una disposición más flexible de los componentes en una cuadrícula que se puede ajustar de acuerdo a las necesidades específicas del diseño. A diferencia de Grid Layout, permite que los componentes ocupen múltiples celdas y tengan tamaños variables.

Null Layout

Podemos decidir no utilizar Layout para ubicar los componentes de la ventana, en este caso el Layout se establece a null y se decide la posición de cada botón y qué tamaño ocupa bien en el código o en el entorno visual. Si se expande la ventana los componentes seguirán en su sitio, no se expandirán con la ventana.

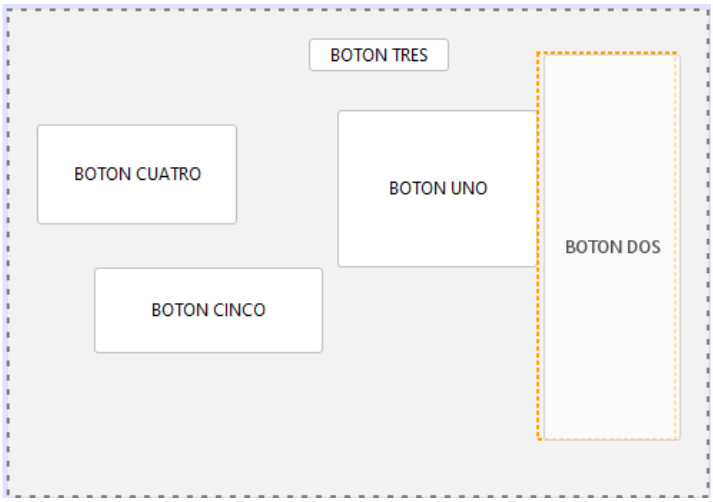


Ilustración 19. Distribución Null Layout

Overlay Layout

Permite superponer varios componentes en el mismo espacio del contenedor. Este diseño resulta útil cuando se necesita colocar elementos visuales, como imágenes con botones o textos informativos sobre un fondo.

Un escenario relevante sería en una aplicación de galerías de imágenes donde se desea mostrar una foto de fondo y, sobre ella, los botones de "Siguiente" y "Anterior". Con Overlay Layout, los botones pueden posicionarse sobre la imagen en el área deseada, asegurando que sean accesibles sin obstruir la vista de la imagen de fondo.

8. 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.

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.

8.1. CLASES

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

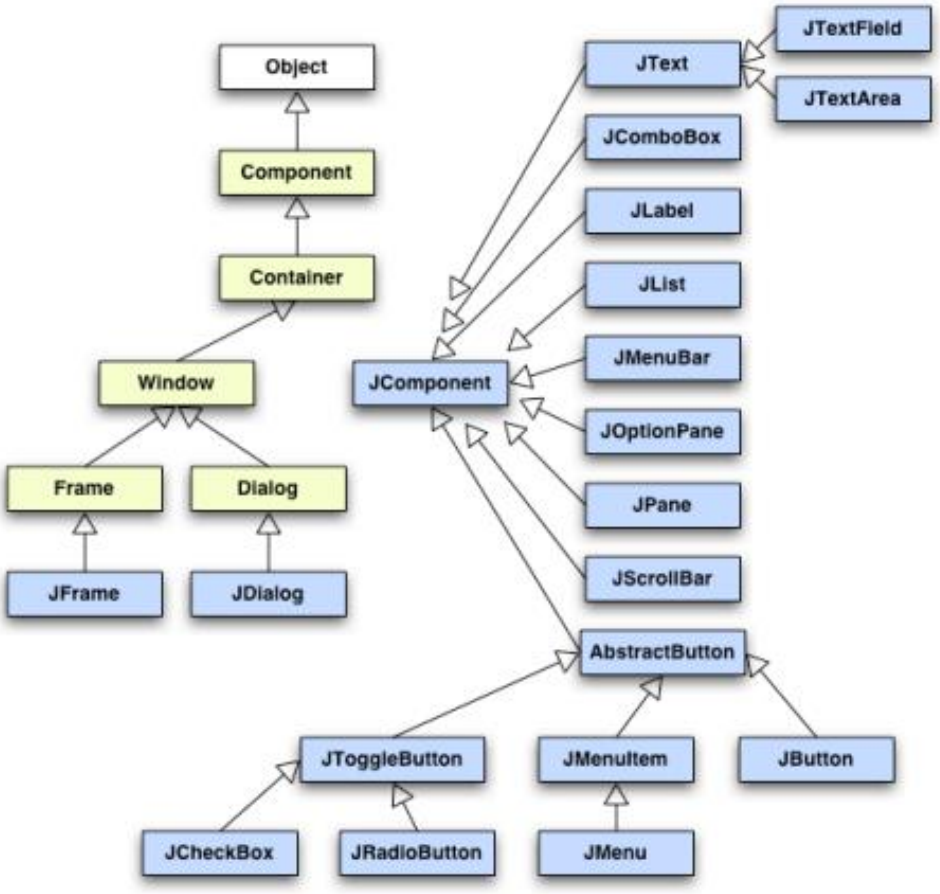


Ilustración 20. Jerarquía de clases de Java Swing

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 un cuadro de diálogo.
- `JApplet` permite crear un **applet swing**.

Para dar funcionalidad a las ventanas, 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.

8.2. 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 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.

8.3. 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 sus propios métodos, mediante los cuales podemos comunicarnos con él en tiempo de ejecución, permitiendo adaptar su apariencia y comportamiento.

9. EVENTOS

Los eventos son acciones o sucesos que se producen en un entorno de programación, a menudo como resultado de la interacción de un usuario con la interfaz de la aplicación. Estas interacciones pueden incluir pulsaciones de teclas, clics del mouse, desplazamientos, y cambios en campos de formulario, entre otros. En el ámbito del desarrollo de interfaces de usuario, los eventos permiten que la aplicación responda a las acciones del usuario, generando una experiencia interactiva.

Cuando un evento se activa, puede ser capturado y manejado, lo que provoca la ejecución de funciones específicas en respuesta a esa acción. La gestión de eventos implica varios pasos, que van desde la detección de que ha ocurrido un evento hasta la ejecución del código necesario para abordarlo.

Cada evento puede transportar información relevante que ofrece datos sobre la interacción, como la ubicación del cursor en un clic o el valor alternado de un campo de formulario cuando se modifica. Esta información permite ajustar el flujo de la aplicación de acuerdo con las necesidades del usuario, mejorando la usabilidad y la efectividad funcional.

Asimismo, la gestión adecuada de eventos influye en el rendimiento de la aplicación. Un manejo ineficaz puede provocar problemas como la falta de respuesta de la interfaz, tiempos de carga prolongados, o errores en el funcionamiento general. Por esto, se deben adoptar buenas prácticas al implementar la lógica de eventos, garantizando que las reacciones sean rápidas y efectivas.

Normalmente, llamamos evento a cualquier interacción que realiza el usuario con la aplicación, como puede ser pulsar un botón con el ratón. hacer doble clic, pulsar y arrastrar, pulsar una combinación de teclas en el teclado, etc.

9.1. PROGRAMACIÓN DIRIGIDA POR EVENTOS

La programación guiada por eventos es un paradigma en el que la estructura y la ejecución de un programa dependen de los eventos que ocurren en el sistema. Estos eventos pueden ser definidos por el usuario o generados por el propio sistema.

Para comprender este enfoque, es útil compararlo con la programación secuencial o estructurada. En esta última, el programador establece de antemano el flujo del programa. En cambio, en la programación guiada por eventos, el flujo es determinado por el usuario u otras acciones que desencadenan los eventos, lo que la convierte en la base de las interfaces de usuario.

La programación orientada a eventos permite que el usuario interactúe con el programa en cualquier momento de su ejecución. Esto se logra a través de un bucle externo permanente que monitorea los eventos y activa los procesos correspondientes para gestionarlos. Normalmente, este bucle está oculto al programador, quien solo debe enfocarse en manejar los eventos. Sin

embargo, en algunos entornos de desarrollo (IDE), es necesario que el programador lo construya explícitamente.

9.2. MODELO DE GESTIÓN DE EVENTOS

En la actualidad, la mayoría de los sistemas operativos emplean interfaces gráficas de usuario, las cuales están en constante vigilancia del entorno para capturar y procesar los eventos que se generan.

El sistema operativo notifica estos eventos a los programas en ejecución, y cada programa, según su programación, decide cómo responder a ellos. Por ejemplo, cada vez que un usuario realiza una acción en una aplicación programada en Java, como hacer clic con el ratón o presionar una tecla, se genera un evento que el sistema operativo comunica a Java.

Java, a su vez, crea un objeto de una clase específica de evento y lo envía a un método particular para que lo gestione. El modelo de eventos en Java se basa en la delegación, lo que significa que la responsabilidad de manejar un evento que ocurre en un objeto fuente recae en otro objeto oyente.

Las fuentes de eventos, o "event sources", son objetos que detectan eventos y notifican a los receptores cuando estos se producen. Ejemplos de fuentes de eventos incluyen un botón que se presiona, un campo de texto que pierde el foco, una tecla presionada en un campo de texto, o una ventana que se cierra, entre otros.

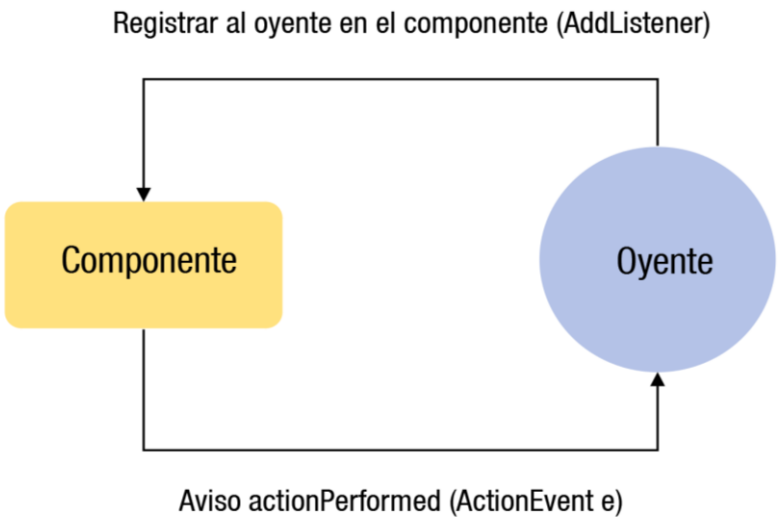


Ilustración 21. Programación guida por eventos

9.3. EVENTOS EN JAVA SWING

Los eventos en Java se organizan en una jerarquía de clases que podemos examinar en la siguiente imagen:

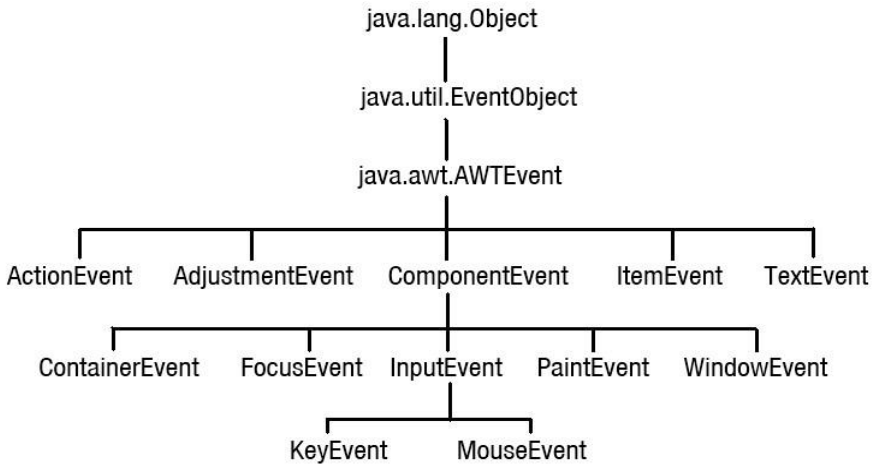


Ilustración 22. Jerarquía de clases asociadas a los diferentes eventos en Java Swing

9.4. EVENTOS DE TECLADO

Los eventos de teclado se generan como respuesta a que el usuario pulsa o libera una tecla mientras un componente tiene el foco de entrada.

`KeyListener` (oyente de teclas)	
Método	**Causa de la invocación**
`keyPressed (KeyEvent e)`	Se ha pulsado una tecla.
`keyReleased (KeyEvent e)`	Se ha liberado una tecla.
`keyTyped (KeyEvent e)`	Se ha pulsado (y a veces soltado) una tecla.
`KeyEvent` (evento de teclas)	
Métodos más usuales	**Explicación**
`char getKeyChar()`	Devuelve el carácter asociado con la tecla pulsada.
`int getKeyCode()`	Devuelve el valor entero que representa la tecla pulsada.
`String getKeyText()`	Devuelve un texto que representa el código de la tecla.
`Object getSource()`	Método perteneciente a la clase `EventObject`. Indica el objeto que produjo el evento.

La clase `KeyEvent`, define muchas constantes así, por ejemplo, `KeyEventVK_A` especifica la tecla A y `KeyEventVK_ESCAPE` especifica la tecla ESCAPE.

9.5. EVENTOS DE RATÓN

Similarmente a los eventos de teclado, los eventos del ratón se generan como respuesta a que el usuario pulsa o libera un botón del ratón, o lo mueve sobre un componente.

`MouseListener` (oyente de ratón)	
Método	**Causa de la invocación**
`mousePressed (MouseEvent e)`	Se ha pulsado un botón del ratón en un componente.
`mouseReleased (MouseEvent e)`	Se ha liberado un botón del ratón en un componente.
`mouseClickerd (MouseEvent e)`	Se ha pulsado y liberado un botón del ratón sobre un componente.
`mouseEntered (KeyEvent e)`	Se ha entrado (con el puntero del ratón) en un componente.
`mouseExited (KeyEvent e)`	Se ha salido (con el puntero del ratón) de un componente.

`MouseMotionListener` (oyente de ratón)	
Método	**Causa de la invocación**
`mouseDragged (MouseEvent e)`	Se presiona un botón y se arrastra el ratón.
`mouseMoved (MouseEvent e)`	Se mueve el puntero del ratón sobre un componente.

`MouseWheelListener` (oyente de ratón)	
Método	**Causa de la invocación**
`MouseWheelMoved (MouseWheelEvent e)`	Se mueve la rueda del ratón.

9.6. OTROS EVENTOS DE JAVA SWING

Mostramos a continuación la relación d otros eventos que podemos utilizar al programar interfaces en el entorno de Java Swing.

ActionEvent

Se genera cuando se lleva a cabo una acción, como hacer clic en un botón, seleccionar un elemento de un menú o presionar la tecla Enter en un campo de texto.

Método manejador típico:

- actionPerformed(ActionEvent e)

WindowEvent

Se genera en respuesta a cambios en el estado de una ventana, como abrir, cerrar, minimizar, maximizar o enfocar la ventana.

Métodos manejadores típicos:

- windowOpened(WindowEvent e)
- windowClosing(WindowEvent e)
- windowClosed(WindowEvent e)
- windowIconified(WindowEvent e)
- windowDeiconified(WindowEvent e)
- windowActivated(WindowEvent e)
- windowDeactivated(WindowEvent e)

FocusEvent

Se genera cuando un componente gana o pierde el foco, es decir, cuando un componente está listo para recibir la entrada del teclado.

Métodos manejadores típicos:

- focusGained(FocusEvent e)
- focusLost(FocusEvent e)

ItemEvent

Se genera cuando cambia el estado de un elemento que es parte de un grupo de selección, como una casilla de verificación o un botón de opción.

Método manejador típico:

- itemStateChanged(ItemEvent e)

ComponentEvent

Se genera cuando un componente es movido, redimensionado, oculto o mostrado.

Métodos manejadores típicos:

- componentResized(ComponentEvent e)
- componentMoved(ComponentEvent e)
- componentShown(ComponentEvent e)
- componentHidden(ComponentEvent e)

ContainerEvent

Se genera cuando un componente es agregado o removido de un contenedor.

Métodos manejadores típicos:

- componentAdded(ContainerEvent e)
- componentRemoved(ContainerEvent e)

AdjustmentEvent

Se genera cuando cambia el valor de un componente ajustable, como una barra de desplazamiento.

Método manejador típico:

- adjustmentValueChanged(AdjustmentEvent e)

TextEvent

Se genera cuando cambia el valor de un texto en un componente de texto, como un `TextField` o un `TextArea`.

Método manejador típico:

- textValueChanged(TextEvent e)

InputMethodEvent

Se genera cuando cambia el método de entrada, especialmente en aplicaciones que manejan lenguajes que requieren un método de entrada complejo.

Métodos manejadores típicos:

- inputMethodTextChanged(InputMethodEvent e)

- caretPositionChanged(InputMethodEvent e)

9.7. ASOCIAR EVENTOS A COMPONENTES EN NETBEANS

Una vez incorporado un componente a nuestra interfaz, podemos asociarle un evento de un modo sencillo accediendo al menú contextual que aparece al pulsar el botón derecho del ratón sobre el mismo:

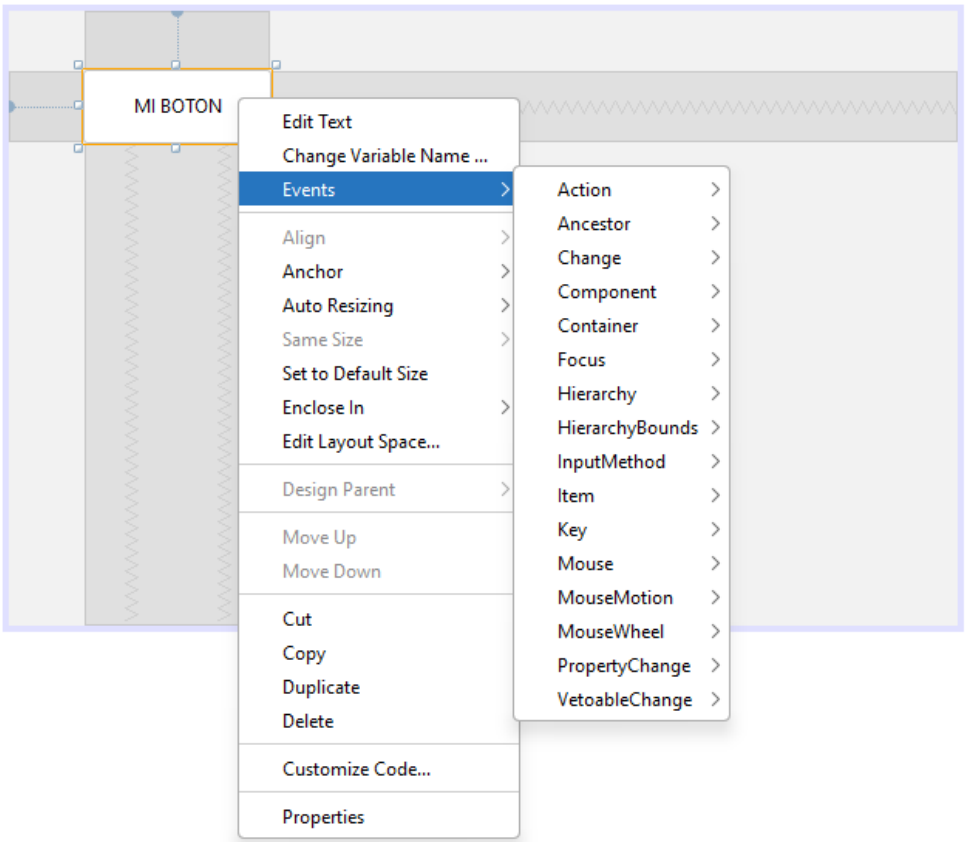


Ilustración 23. Asociación de eventos a componente o control

Seleccionando uno de los tipos de evento se generará el consiguiente código de un método en el que podremos definir nuestras acciones asociadas al evento.

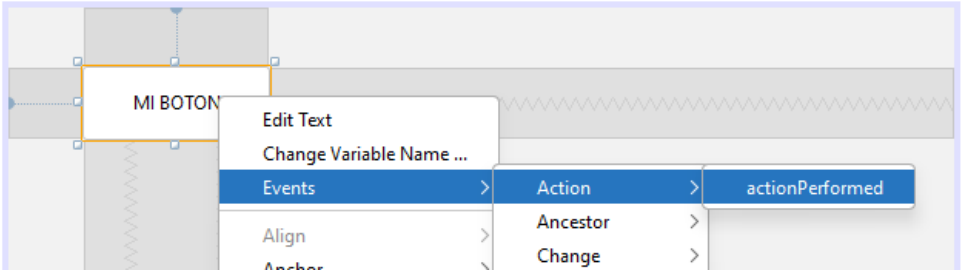


Ilustración 24. Evento actionPerformed asociado al botón

En este caso lanzaremos un JOptionPane con un mensaje:

```
private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    JOptionPane.showMessageDialog(this, "Ventana llamada desde MI BOTÓN");
}
```

Ilustración 25. Método generado al asociar el evento al botón

Los resultados al pulsar el botón se muestran en la siguiente imagen:

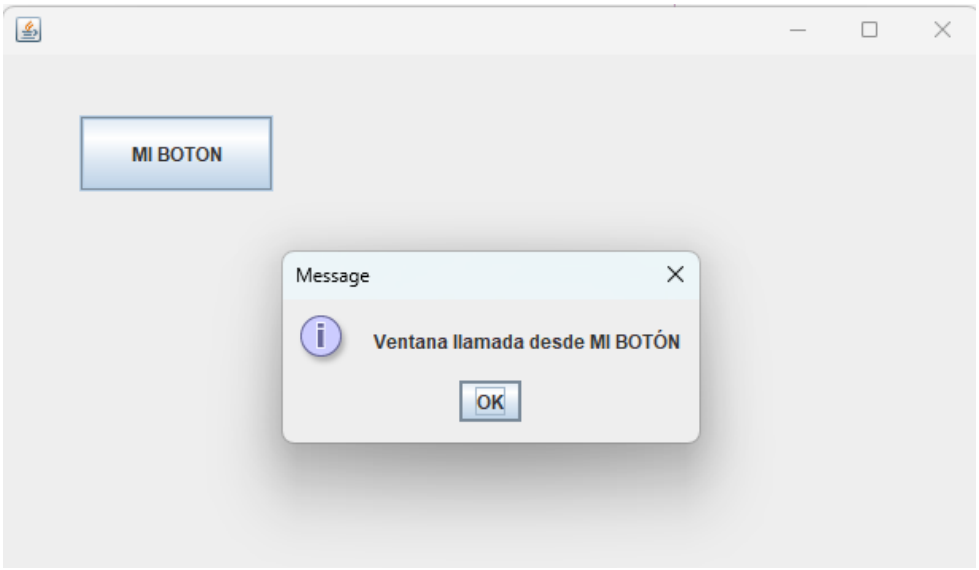


Ilustración 26. Resultado de la acción del pulsar el botón

10. DIÁLOGOS MODALES Y NO MODALES

10.1. INTRODUCCIÓN

Los diálogos modales y no modales son componentes importantes en la interacción con el usuario dentro de software. Estos elementos permiten gestionar la comunicación entre el sistema y el usuario, ofreciendo una experiencia fluida y eficiente. A continuación, se presentan un análisis detallado de ambos tipos de diálogos con ejemplos y aplicaciones relevantes para cada uno.

Los diálogos modales son aquellos que bloquean la interacción con la interfaz principal hasta que el usuario toma una decisión respecto al contenido presentado en el diálogo. Este tipo de ventanas es útil en situaciones que requieren atención inmediata y pueden influir en el flujo de la aplicación. Un ejemplo común es el diálogo que se muestra al intentar cerrar un documento sin guardar cambios. En esta situación, se puede presentar un mensaje que diga: "¿Está seguro de que desea cerrar sin guardar?".

Otro caso de uso de diálogo modal se encuentra en el proceso de creación de cuentas en un sitio web. Si un usuario intenta registrarse con un nombre de usuario en uso, puede aparecer un aviso modal que indique "Este nombre de usuario ya está en uso, por favor elija otro". Este mensaje requiere que el usuario interactúe con él antes de continuar con el registro.

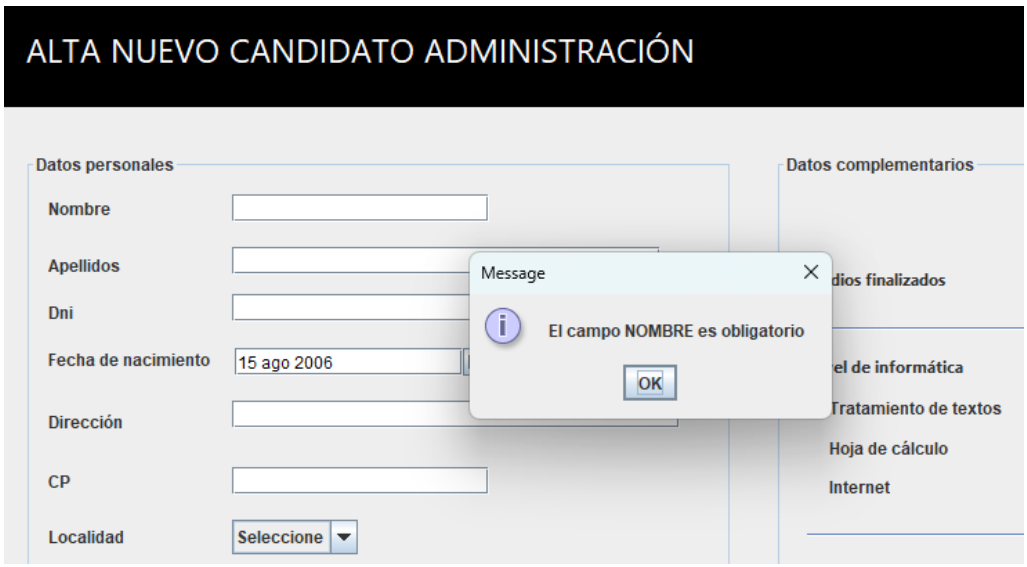


Ilustración 27. Ventana modal generada a partir de un JOptionPane

Además, estas ventanas se utilizan también para advertir sobre errores significativos. Por ejemplo, en un sistema de gestión de proyectos, si un usuario intenta eliminar un proyecto que contiene tareas asignadas, se puede mostrar un diálogo que advierte sobre la posible pérdida de datos. Este enfoque evita acciones involuntarias y fortalece el control del usuario sobre sus decisiones.

Por el contrario, los diálogos no modales permiten que los usuarios interactúen con otras partes de la aplicación mientras el diálogo permanece abierto. Este tipo de ventana es adecuado para

proporcionar información adicional o rutas de navegación que no interrumpen el trabajo principal. Un ejemplo sería un cuadro de preferencias en una aplicación de edición de gráficos. Este cuadro podría mantenerse abierto mientras el usuario trabaja en su diseño, permitiendo ajustes en configuraciones de color, tamaño o filtros sin interrumpir su proceso de creación.

La selección adecuada entre diálogos modales y no modales en el desarrollo de interfaces impacta significativamente la experiencia del usuario. La elección entre uno y otro depende de la situación y la naturaleza de la interacción deseada, así como de los objetivos de la aplicación. La implementación cuidadosa de estos componentes, junto con pruebas de usabilidad y una atención a la accesibilidad, permite construir aplicaciones efectivas y más amigables para los usuarios.

10.2. DIÁLOGOS MODALES

Los diálogos modales se utilizan de forma generalizada en aplicaciones y por el propio sistema operativo.

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);
   }
}
```

A partir de Java 6, 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`.

10.3. 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.

Ilustración 28. Diálogo no modal

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.

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);
  }
}
```

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

11. CUADROS DE DIÁLOGO CON JOPTIONPANE

Los cuadros de diálogo son una opción perfecta para enviar al usuario mensajes de todo tipo: de error, de precaución, de información, etc. Además, permiten mostrar uno o más botones, e incluso solicitar información al usuario en forma de cuadro de texto o combo.

Para crear cuadros de diálogo utilizamos el objeto JOptionPane, y en función del método a elegir, mostrará un tipo u otro. Veamos los métodos más utilizados:

showMessageDialog

es el más habitual, y despliega un cuadro de diálogo con un único botón (Aceptar). Tiene varias sobrecargas del método, es decir, podemos elegir el mismo método con dos, cuatro o cinco parámetros, que se indican a continuación:

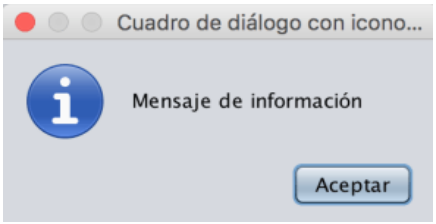


Ilustración 29. showMessageDialog

- Con dos parámetros:

```
JOptionPane.showMessageDialog(Componente padre, mensaje);
```

- Con cuatro parámetros:

```
JOptionPane.showMessageDialog(Componente padre, mensaje,
título, tipo de mensaje);
```

- Con cinco parámetros:

```
JOptionPane.showMessageDialog(Componente padre, mensaje,
título, tipo de mensaje, icono);
```

Información sobre los diferentes tipos de parámetros:

- Componente sobre el que se abre el cuadro de diálogo (lo dejamos por defecto).
- Mensaje que se mostrará en el cuadro de diálogo.
- Título de la ventana del cuadro de diálogo.
- Entero que indica qué tipo de icono se va a mostrar, y puede ser:
  - JOptionPane.DEFAULT_OPTION: no muestra icono (como PLAIN_MESSAGE)
  - JOptionPane.INFORMATION_MESSAGE
  - JOptionPane.WARNING_MESSAGE
  - JOptionPane.ERROR_MESSAGE

- JOptionPane.QUESTION_MESSAGE
- JOptionPane.PLAIN_MESSAGE: sin icono.

showConfirmDialog

Por defecto pide al usuario que confirme entre las opciones Sí y No, pero puede ser configurado para adaptarse a las necesidades del programador. Dispone de cuatro sobrecargas del método donde podemos elegir los siguientes parámetros:

```
JOptionPane.showConfirmDialog(Componente padre, mensaje,
titulo, tipo de seleccion, tipo de mensaje, icono);
```

- Componente sobre el que se abre el cuadro de diálogo (lo dejamos por defecto)
- Mensaje que se mostrará en el cuadro de diálogo.
- Título de la ventana del cuadro de diálogo.
- Tipo de selección, que puede ser:
  - JOptionPane.OK_CANCEL_OPTION: con dos botones.
  - JOptionPane.YES_NO_OPTION: con dos botones.
  - JOptionPane.YES_NO_CANCEL_OPTION: con tres botones.
- Entero que indica qué tipo de icono se va a mostrar (ver showMessageDialog).
- Icono a elección del usuario.

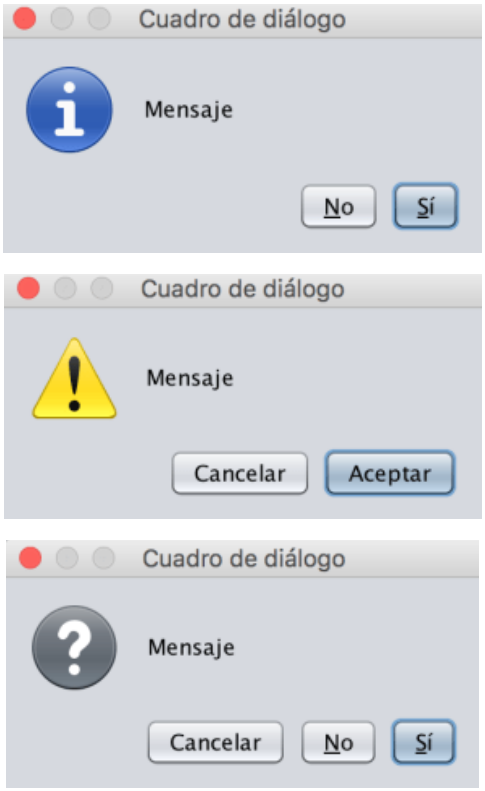


Ilustración 30. showConfirmDialog

showOptionDialog

Despliega un cuadro de diálogo diseñado por el programador y tiene los siguientes parámetros:

```
JOptionPane.showOptionDialog(padre, mensaje, titulo, tipo de
seleccion, tipo de mensaje, icono, opciones, valor inicial);
```

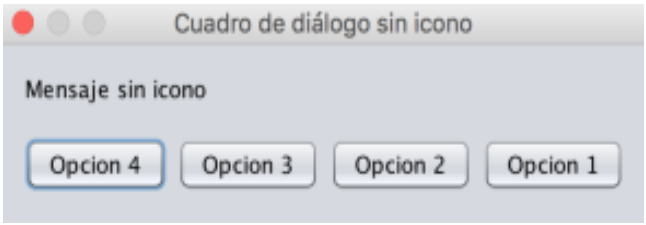


Ilustración 31. showOptionDialog

- Componente sobre el que se abre el cuadro de diálogo (lo dejamos por defecto)
- Mensaje que se mostrará en el cuadro de diálogo.
- Título de la ventana del cuadro de diálogo.
- Tipo de selección, utilizamos por defecto JOptionPane.DEFAULT_OPTION.
- Entero que indica qué tipo de icono se va a mostrar (ver showMessageDialog).
- Icono a elección del usuario.
- Opciones: le paso un array de cadenas que almacenarán el texto de los botones a mostrar.
- Valor inicial: el botón que se encuentra marcado por defecto.

showInputDialog

despliega un cuadro de diálogo que puede almacenar, a su vez, un cuadro de texto, un combo o una lista. Veamos las diferentes opciones:

```
JOptionPane.showInputDialog(Componente padre, Mensaje,
Titulo, Tipo De Mensaje, Icono, Array de opciones, Seleccion
por defecto);
```

Como se puede observar, tiene las mismas opciones que en el caso anterior. Vamos a ver cómo poder utilizar cada uno tipos.

- Solamente un cuadro de texto (el texto por defecto es opcional).

  ```
  JOptionPane.showInputDialog(Componente padre, mensaje,
  texto por defecto);
  ```

- Un cuadro de texto con las opciones habituales:

  ```
  JOptionPane.showInputDialog(Componente padre, Mensaje,
  Titulo, Tipo de mensaje);
  ```

- Un combo desplegable con una opción marcada por defecto:

```
JOptionPane.showInputDialog(padre, mensaje, titulo, tipo de
seleccion, tipo de mensaje, icono, opciones, valor
inicial);
```

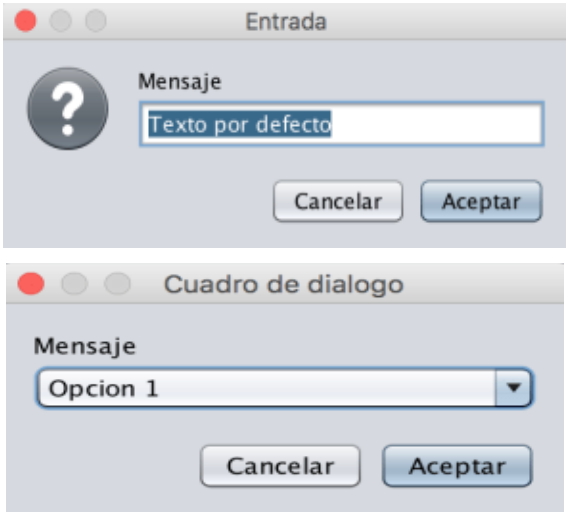


Ilustración 32. showInputDialog

No hay que olvidar que cuando un cuadro de diálogo devuelve un entero, sus posibles valores son:

- JOptionPane.YES_OPTION
- JOptionPane.NO_OPTION
- JOptionPane.CANCEL_OPTION
- JOptionPane.OK_OPTION
- JOptionPane.CLOSED_OPTION: el usuario cierra la ventana.

Por tanto, habría que capturar la opción introducida por el usuario, por ejemplo, de la siguiente manera:

```
int opcion = JOptionPane.showXXXDialog(<parametros>);
if (opcion==JOptionPane.YES_OPTION) { ...}
```

RESUMEN

La creación de interfaces de usuario es crucial en el desarrollo de aplicaciones, ya que estas interfaces impactan en la experiencia de los usuarios, influyendo en su satisfacción y efectividad al interactuar con el software. Para facilitar este proceso, se dispone de una variedad de librerías de componentes que se adaptan a diferentes sistemas operativos y lenguajes de programación. Estas librerías proporcionan herramientas y elementos visuales predefinidos que los desarrolladores pueden utilizar para crear interfaces de manera más eficiente. Librerías como React, Xamarin y JavaFX son ejemplos representativos de cómo diferentes plataformas ofrecen recursos que mejoran la productividad y calidad del diseño de la interfaz.

El uso de herramientas propietarias y de código abierto es una práctica común en la creación de interfaces. Herramientas propietarias como Adobe XD o Microsoft Visual Studio ofrecen características robustas y soporte técnico frecuente, mientras que herramientas de código abierto como Figma o GIMP permiten una mayor flexibilidad y personalización, fomentando la colaboración y el acceso a recursos comunitarios.

Diferentes entornos de desarrollo integrados (IDEs) desempeñan un rol significativo en el diseño y creación de interfaces de usuario. Plataformas como IntelliJ IDEA, Visual Studio y Android Studio no solo proporcionan editores de código, sino que también integran herramientas que optimizan el flujo de trabajo del desarrollador. Estas plataformas permiten a los desarrolladores arrastrar y soltar componentes en áreas de diseño, facilitando la creación de layouts de manera intuitiva y eficiente. Eclipse, Visual Studio y Xcode son ejemplos destacados de IDEs que proporcionan funcionalidades avanzadas para la creación y gestión de interfaces.

El desarrollo de interfaces también se beneficia del uso de contenedores, que organizan y estructuran los componentes visuales dentro de una aplicación. Los contenedores mantienen una jerarquía clara de información y la presentan de manera adaptativa en diferentes dispositivos, lo cual es crucial en entornos multiplataforma. Variados tipos de contenedores como paneles y diseños de cuadrícula facilitan la administración eficiente y mejora la experiencia del usuario al asegurar una alineación correcta y un flujo de contenido adecuado.

La adición y eliminación de componentes en la interfaz se realiza de manera controlada para mantener una disposición accesible y lógica para el usuario final. La ubicación y alineación adecuada de estos componentes influye en cómo los usuarios interactúan con la aplicación. La apariencia y la funcionalidad resultan de la organización de los elementos dentro de la interfaz.

En relación a la modificación de propiedades de los componentes, es esencial para personalizar la experiencia del usuario y adaptar los elementos visuales a las necesidades de la aplicación. Esto incluye la asociación de acciones a eventos, asegurando que los componentes respondan adecuadamente a las interacciones del usuario. Diálogos modales y no modales son utilizados dependiendo de la necesidad de captar la atención del usuario o de brindar flexibilidad mientras se interactúa con la aplicación principal.

Las características de los componentes son determinantes en la interacción y la experiencia del usuario, abarcando aspectos como interactividad, presentación visual, accesibilidad,

consistencia, adaptabilidad, personalización y retroalimentación. La interactividad incluye componentes que reaccionan a las acciones del usuario, mejorando la usabilidad de la herramienta. La presentación visual y la accesibilidad aseguran que el usuario pueda utilizar la interfaz de manera eficiente y agradable. La consistencia en diseño facilita la navegación, mientras que la adaptabilidad garantiza que la interfaz sea utilizable en distintos dispositivos. La personalización permite a los usuarios ajustar la interfaz a sus preferencias, mejorando su satisfacción.

Para concluir, la creación de interfaces de usuario incluye una serie de prácticas y consideraciones desde la selección de herramientas y librerías hasta la organización y personalización de componentes. Estas prácticas aseguran que las interfaces sean eficaces, accesibles y proporcionen una experiencia satisfactoria a los usuarios finales.