

Comunicándonos con el usuario. Interfaces gráficas.

Caso práctico



Ana está cursando el módulo de Programación.

En el aula suele sentarse junto a su compañero **José Javier**.

En clase llevan unos días explicándoles cómo construir aplicaciones Java, utilizando interfaces gráficas de usuario (GUI).

Pero, ¿qué es una interfaz de usuario? A grandes rasgos, les han comentado que una interfaz de usuario es el medio con que el usuario puede comunicarse con una computadora. Las interfaces gráficas de usuario incluyen elementos tales como: menús, ventanas, paneles, etc., en general, elementos gráficos que facilitan la comunicación entre el ser humano y la computadora.

Hasta ahora, las aplicaciones de ejemplo que habían realizado, estaban en modo consola, o modo carácter, y están contentos porque están viendo las posibilidades que se les abren ahora. Están comprobando que podrán dar a sus aplicaciones un aspecto mucho más agradable para el usuario. **Ana** le comenta a **José Javier**:

—Así podremos dar un aspecto profesional a nuestros programas.



Materiales actualizados por el profesorado de la Junta de Andalucía.

[Aviso Legal](#)



Materiales formativos de FP Online propiedad del Ministerio

de Educación, Cultura y Deporte.

[Aviso Legal](#)

1.- Introducción.

Hoy en día las **interfaces** de los programas son cada vez más sofisticadas y atractivas para el usuario. Son intuitivas y cada vez más fáciles de usar: pantallas táctiles, etc.

Sin embargo, no siempre ha sido así. No hace muchos años, antes de que surgieran y se popularizaran las interfaces gráficas para que el usuario interactuara con el sistema operativo con sistemas como Windows, etc., se trabajaba en **modo consola**, o **modo carácter**, es decir, se le daban las ordenes al ordenador con comandos por teclado, de hecho, por entonces no existía el ratón. Es lo que se conoce como **interfaces de línea de comandos (CLI)**.



youflavio

Así que, con el tiempo, con la idea de simplificar el uso de los ordenadores para extender el uso a un cada vez mayor espectro de gente, de usuarios de todo tipo, y no sólo para los expertos, se ha convertido en una práctica habitual utilizar **interfaces gráficas de usuario (IGU ó GUI en inglés)**, para que el usuario interactúe y establezca un contacto más fácil e intuitivo con el ordenador.

En ocasiones verás otras definiciones de interfaz, como la que define una interfaz como un dispositivo que permite comunicar dos sistemas que no hablan el mismo lenguaje. También se emplea el término interfaz para definir el juego de conexiones y dispositivos que hacen posible la comunicación entre dos sistemas.

Aquí en este módulo, cuando hablamos de interfaz nos referimos a la cara visible de los programas tal y como se presenta a los usuarios para que interactúen con la máquina. La interfaz gráfica implica la presencia de un monitor de ordenador, en el que veremos la interfaz constituida por una serie de **menús e iconos que representan las opciones que el usuario** puede tomar dentro del sistema.

Las interfaces gráficas de usuario proporcionan al usuario **ventanas, cuadros de diálogo, barras de herramientas, botones, listas desplegables** y muchos otros elementos. Las aplicaciones son conducidas por eventos y se desarrollan haciendo uso de las clases que para ello nos ofrece el **API** de Java.

En 1981 aparecieron los primeros ordenadores personales, los llamados **PC**, pero hasta 1993 no se generalizaron las interfaces gráficas de usuario. El escritorio del sistema operativo Windows de Microsoft y su sistema de ventanas sobre la pantalla se ha estandarizado y universalizado, pero fueron los ordenadores Macintosh de la compañía Apple los pioneros en introducir las interfaces gráficas de usuario.

Para saber más

En el siguiente enlace puedes ver la evolución en las interfaces gráficas de diverso software, entre ellos, el de las GUI de MAC OS, o de Windows en las sucesivas versiones

[Galería de interfaces gráficas](#)



Autoevaluación

Señala la opción correcta acerca de las interfaces gráficas de usuario:

☐

Son sinónimos de ficheros de texto.

☐

Las ventanas de una aplicación no serían un ejemplo de elemento de interfaz gráfica de usuario.

☐

Surgen con la idea de facilitar la interacción del usuario con la máquina.

☐

Ninguna es correcta.

2.- Librerías de Java para desarrollar GUI.

Caso práctico

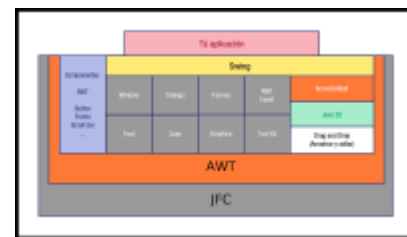


José Javier está un poco abrumado por la cantidad de componentes que tiene Java para desarrollar interfaces gráficas. Piensa que hay tantos, que son tantas clases, que nunca podrá aprendérselos. **Ana** le dice:

—**José Javier**, no te preocupes, seguro que haciendo programas, enseguida ubicarás los que más utilices, y el resto, siempre tienes la documentación de Java para cuando dudes, consultarla.

Hemos visto que la interfaz gráfica de usuario es la parte del programa que permite al usuario interactuar con él. Pero, ¿cómo la podemos crear en Java?

El API de Java proporciona una librería de clases para el desarrollo de interfaces gráficas de usuario (en realidad son dos: AWT y Swing). Esas librerías se engloban bajo los nombres de AWT y Swing, que a su vez forman parte de las **Java Foundation Classes o JFC**.



Entre las clases de las JFC hay un grupo de elementos importante **que ayuda a la construcción de interfaces gráficas de usuario (GUI)** en Java.

Los elementos que componen las JFC son:

- ✔ **Componentes Swing:** encontramos componentes tales como botones, cuadros de texto, ventanas o elementos de menú.
- ✔ **Soporte de diferentes aspectos y comportamientos (Look and Feel):** permite la elección de diferentes apariencias de entorno. Por ejemplo, el mismo programa puede adquirir un aspecto **Metal** Java (multiplataforma, igual en cualquier entorno), **Motif** (el aspecto estándar para entornos Unix) o **Windows** (para entornos Windows). De esta forma, el aspecto de la aplicación puede ser independiente de la plataforma, lo cual está muy bien para un lenguaje que lleva la seña de identidad de multiplataforma, y se puede elegir el que se desee en cada momento.
- ✔ **Interfaz de programación Java 2D:** permite incorporar gráficos en dos dimensiones, texto e imágenes de alta calidad.
- ✔ **Soporte de arrastrar y soltar (Drag and Drop)** entre aplicaciones Java y aplicaciones nativas. Es decir, se implementa un portapapeles. Llamamos aplicaciones nativas a las que están desarrolladas en un entorno y una plataforma concretos (por ejemplo Windows o Unix), y sólo funcionarán en esa plataforma.
- ✔ **Soporte de impresión.**
- ✔ **Soporte sonido:** captura, reproducción y procesamiento de datos de audio y MIDI.
- ✔ **Soporte de dispositivos de entrada distintos del teclado**, para japonés, chino, etc.

- ✔ **Soporte de funciones de Accesibilidad, para crear interfaces para discapacitados:** permite el uso de tecnologías como los lectores de pantallas o las pantallas Braille adaptadas a las personas discapacitadas.

Con estas librerías, Java proporciona un conjunto de herramientas para la construcción de interfaces gráficas que tienen una apariencia y se comportan de forma semejante en todas las plataformas en las que se ejecuten.



Autoevaluación

Señala la opción incorrecta. JFC se consta de los siguientes elementos:

☐

Componentes Swing.

☐

Soporte de diversos "look and feel".

☐

Soporte de impresión.

☐

Interfaz de programación Java 3D.

2.1.- AWT.

Cuando apareció Java, los componentes gráficos disponibles para el desarrollo de GUI se encontraban en la librería denominada Abstract Window Toolkit (AWT).



Las clases AWT se desarrollaron usando código nativo (o sea, código asociado a una plataforma concreta). Eso **dificultaba la portabilidad** de las aplicaciones. Al usar código nativo, para poder conservar la portabilidad era necesario restringir la funcionalidad a los mínimos comunes a todas las plataformas donde se pretendía usar AWT. Como consecuencia, AWT es una librería con una funcionalidad muy pobre. **AWT es adecuada para interfaces gráficas sencillas**, pero no para proyectos más complejos.

La estructura básica de la librería gira en torno a **componentes y contenedores**. Los contenedores contienen componentes y son componentes a su vez, de forma que los eventos pueden tratarse tanto en contenedores como en componentes.

Más tarde, cuando apareció **Java 2** surgió una librería más robusta, versátil y flexible: **Swing**. En cualquier caso, AWT aún sigue estando disponible. De hecho se usa por los componentes de Swing.

Puedes ver la lista de los principales componentes de la clase AWT en el anexo que hay al final de la unidad.

Para saber más

Página oficial de Oracle sobre AWT (en inglés).

[AWT \(Abstract Window Toolkit\)](#)

AWT sigue siendo imprescindible, ya que todos **los componentes Swing se construyen haciendo uso de clases de AWT**. De hecho, como puedes comprobar en el API, todos los componentes Swing, como por ejemplo JButton (es la clase Swing que usamos para crear cualquier botón de acción en una ventana), derivan de la clase JComponent, que a su vez deriva de la clase AWT Container.

Las clases asociadas a cada uno de los **componentes AWT** se encuentran en el paquete java.awt. Las clases relacionadas con el manejo de **eventos** en AWT están en el paquete java.awt.event.

AWT fue la primera forma de construir las ventanas en Java, pero:

- ✔ Limitaba la portabilidad.
- ✔ Restringía la funcionalidad.
- ✔ Requería demasiados recursos.



Autoevaluación

AWT está indicado para proyectos muy grandes y de gran complejidad.

Verdadero

☐

Falso

☐

Para saber más

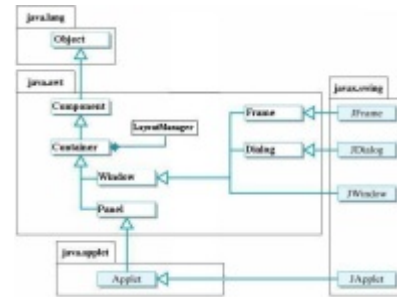
Decíamos más arriba, que las **JFC** proporcionan soporte para impresión. En los siguientes enlaces tienes más información sobre impresión en Java.

[Impresión en Java](#)

[Imprimiendo en Java.](#)

2.2.- Swing.

Cuando se vio que era necesario mejorar las características que ofrecía AWT, distintas empresas empezaron a sacar sus controles propios para mejorar algunas de las características de AWT. Así, Netscape sacó una librería de clases llamada **Internet Foundation Classes** para usar con Java, y eso obligó a Sun (todavía no adquirida por Oracle) a reaccionar para adaptar el lenguaje a las nuevas necesidades.



Se desarrolló en colaboración con Netscape todo el conjunto de componentes Swing que se añadieron a la JFC.

Swing es una librería de Java para la generación del GUI en aplicaciones.

Swing se apoya sobre AWT y añade JComponents. La arquitectura de los componentes de Swing facilita la personalización de apariencia y comportamiento, si lo comparamos con los componentes AWT.

Debes conocer

A continuación tienes una tabla con la lista de los controles Swing más habituales. Debes tener en cuenta que las imágenes están obtenidas eligiendo el aspecto (LookAndFeel) multiplataforma propio de Java.

[Resumen textual alternativo](#)

Por cada componente AWT (excepto Canvas) existe un componente Swing equivalente, cuyo nombre empieza por J, que permite más funcionalidad siendo menos pesado. Así, por ejemplo, para el componente AWT Button existe el equivalente Swing JButton, que permite como funcionalidad adicional la de crear botones con distintas formas (rectangulares, circulares, etc), incluir imágenes en el botón, tener distintas representaciones para un mismo botón según esté seleccionado, o bajo el cursor, etc.

La razón por la que no existe JCanvas es que los paneles de la clase JPanel ya soportan todo lo que el componente Canvas de AWT soportaba. No se consideró necesario añadir un componente Swing JCanvas por separado.

Algunas otras características más de Swing son:

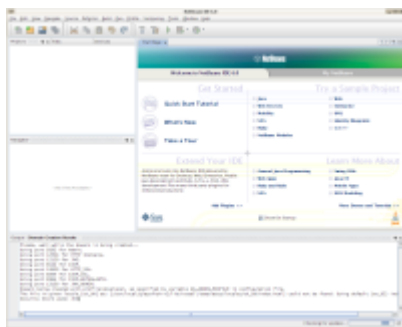
- ✓ Es **independiente de la arquitectura** (metodología no nativa propia de Java)
- ✓ Proporciona **todo lo necesario para la creación de entornos gráficos**, tales como diseño de menús, botones, cuadros de texto, manipulación de eventos, etc.
- ✓ **Los componentes Swing no necesitan una ventana propia del sistema operativo cada uno**, sino que son visualizados dentro de la ventana que los contiene mediante métodos gráficos, por lo que requieren bastantes menos recursos.
- ✓ **Las clases Swing están completamente escritas en Java**, con lo que la **portabilidad es total**, a la vez que no hay obligación de restringir la funcionalidad a los mínimos comunes de

todas las plataformas

- ✓ Las clases Swing aportan una considerable gama de funciones que haciendo uso de la funcionalidad básica propia de AWT **aumentan las posibilidades de diseño** de interfaces gráficas.
- ✓ Debido a sus características, los componentes AWT se llaman componentes “**de peso pesado**” por la gran cantidad de recursos del sistema que usan, y los componentes Swing se llaman componentes “**de peso ligero**” por no necesitar su propia ventana del sistema operativo y por tanto consumir muchos menos recursos.
- ✓ Aunque todos los componentes Swing derivan de componentes AWT y de hecho **se pueden mezclar en una misma aplicación componentes de ambos tipos**, se desaconseja hacerlo. **Es preferible desarrollar aplicaciones enteramente Swing**, que requieren menos recursos y son más portables.

3.- Creación de interfaces gráficas de usuario utilizando asistentes y herramientas del entorno integrado.

Caso práctico



Ilya CDDL

Ana le ha tomado el gusto a hacer programas con la librería Swing de Java y utilizando el IDE NetBeans. Le parece tan fácil que no lo puede creer. Pensaba que sería difícil el uso de los controles de las librerías, pero ha descubierto que es poco menos que cogerlos de la paleta y arrastrarlos hasta el formulario donde quiere situarlos.

—Pero si esto lo podría hacer hasta mi sobrino pequeño,... — piensa para sí.

Crear aplicaciones que incluyan interfaces gráficas de usuario, con NetBeans, es muy sencillo debido a las facilidades que nos proporcionan sus **asistentes** y **herramientas**.

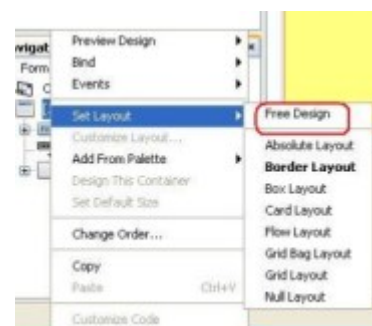
El IDE de programación nos proporciona un diseñador para crear aplicaciones de una manera fácil, sin tener que preocuparnos demasiado del código. Simplemente nos debemos centrar en el funcionamiento de las mismas.

Un programador experimentado, probablemente no utilizará los asistentes, sino que escribirá, casi siempre, todo el código de la aplicación. De este modo, podrá indicar por código la ubicación de los diferentes controles, de su interfaz gráfica, en el contenedor (panel, marco, etc.) en el que se ubiquen.

Pero en el caso de programadores novatos, o simplemente si queremos ahorrar tiempo y esfuerzo, tenemos la opción de aprovecharnos de las capacidades que nos brinda un entorno de desarrollo visual para diseñar e implementar formularios gráficos.

Algunas de las características de NetBeans, que ayudan a la generación rápida de aplicaciones, y en particular de interfaces son:

- ✓ **Modo de diseño libre (Free Design):** permite mover libremente los componentes de interfaz de usuario sobre el panel o marco, sin atenerse a uno de los layouts por defecto.
- ✓ **Independencia de plataforma:** diseñando de la manera más fácil, es decir, arrastrando y soltando componentes desde la paleta al área de diseño visual, el NetBeans sugiere alineación, posición, y dimensión de los componentes, de manera que se ajusten para cualquier plataforma, y en tiempo de ejecución el resultado sea el óptimo, sin importar el sistema



operativo donde se ejecute la aplicación.

- ✔ **Soporte de internacionalización.** Se pueden internacionalizar las aplicaciones Swing, aportando las traducciones de cadenas de caracteres, imágenes, etc., sin necesidad de tener que reconstruir el proyecto, sin tener que compilarlo según el país al que vaya dirigido. Se puede utilizar esta características empleando ficheros de recursos (ResourceBundle files). En ellos, deberíamos aportar todos los textos visibles, como el texto de etiquetas, campos de texto, botones, etc. NetBeans proporciona una asistente de internacionalización desde el menú de herramientas (Tools).



Autoevaluación

NetBeans ayuda al programador de modo que pueda concentrarse en implementar la lógica de negocio que se tenga que ejecutar por un evento dado.

Verdadero ☐ Falso ☐

3.1.- Diseñando con asistentes.

Los pasos para crear y ejecutar aplicaciones, se pueden resumir en:

- ✓ Crear un proyecto.
- ✓ Construir la interfaz con el usuario.
- ✓ Añadir funcionalidad, en base a la lógica de negocio que se requiera.
- ✓ Ejecutar el programa.



Veamos un ejemplo con estos pasos:

- ✓ Primero, crea el proyecto de la manera tan simple como puedes ver en:

[Resumen textual alternativo](#)

- ✓ A continuación, crea la interfaz de usuario tal y como ves en:

[Resumen textual alternativo](#)

- ✓ Ahora, añade la funcionalidad como en:

[Resumen textual alternativo](#)

- ✓ Por último, ejecuta el proyecto:

[Resumen textual alternativo](#)

Es posible que te estés empezando a dar cuenta de cómo se gestionan los eventos en Java. Vamos a analizar un poco más, en la siguiente presentación, cómo funciona la gestión de eventos, al hilo del proyecto que acabamos de crear:

[Resumen textual alternativo](#)

Debes conocer

En el siguiente enlace puedes ver, paso a paso, la creación de un proyecto, que utiliza interfaz gráfica de usuario, con NetBeans

[Proyecto para crear un interfaz gráfico paso a paso](#) (644 KB)

En este tienes un documento muy interesante, paso a paso, sobre construcción de interfaces gráficos con NetBeans.

[Construcción de interfaces gráficos de usuario con NetBeans](#) (399 KB)

Para saber más

En el siguiente vídeo puedes ver el primero de una serie de cinco vídeos en los que se realiza una calculadora con la ayuda de los asistentes de NetBeans.

[Resumen textual alternativo](#)

4.- Eventos.

Caso práctico



Ana sabe que entender cómo funciona la programación por eventos es algo relativamente fácil, el problema está en utilizar correctamente los eventos más adecuados en cada momento. **Ana** tiene muy claro que el evento se asocia a un botón cuando se pulsa, pero **José Javier** la hace dudar, cuando la llama por teléfono para preguntarle unas dudas y le dice, que él cree, que el evento se produce cuando el

botón se suelta. Además, le recuerda que en clase dijeron que al ser enfocado con el ratón, o al accionar una combinación de teclas asociadas, también se pueden producir eventos. Tras hablarlo, piensan que realmente no es tan complicado, porque se repiten muchos eventos y si nos paramos a pensarlo, todos ellos son predecibles y bastante lógicos.

4.1.- Introducción.

¿Qué es un **evento**?

Es todo hecho que ocurre mientras se ejecuta la aplicación. 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;
- ✓ pasar el ratón por encima de un componente;
- ✓ salir el puntero de ratón de un componente;
- ✓ abrir una ventana;
- ✓ etc..



RobertG NL

¿Qué es la **programación guiada por eventos**?

Imagina la ventana de cualquier aplicación, por ejemplo la de un procesador de textos. En esa ventana aparecen multitud de elementos gráficos interactivos, de forma que no es posible que el programador haya previsto todas las posibles entradas que se pueden producir por parte del usuario en cada momento.

Con el control de flujo de programa de la **programación imperativa**, el programador tendría que estar continuamente leyendo las entradas (de teclado, o ratón, etc.) y comprobar para cada entrada o interacción producida por el usuario, de cual se trata de entre todas las posibles, usando estructuras de flujo condicional (if-then-else, **switch**) para ejecutar el código conveniente en cada caso. Si piensas que para cada opción del menú, para cada botón o etiqueta, para cada lista desplegable, y por tanto para cada componente de la ventana, incluyendo la propia ventana, habría que comprobar todos y cada uno de los eventos posibles, nos damos cuenta de que las posibilidades son casi infinitas, y desde luego impredecibles. Por tanto, de ese modo es imposible solucionar el problema.

Para abordar el problema de tratar correctamente las interacciones del usuario con la interfaz gráfica de la aplicación hay que cambiar de estrategia, y la **programación guiada por eventos es una buena solución**. Veamos cómo funciona el modelo de gestión de eventos.

4.2.- Modelo de gestión de eventos.

¿Qué sistema operativo utilizas? ¿Posee un entorno gráfico? Hoy en día, la mayoría de sistemas operativos utilizan interfaces gráficas de usuario. Este tipo de sistemas operativos están **continuamente monitorizando el entorno para capturar y tratar los eventos** que se producen.

El sistema operativo informa de estos eventos a los programas que se están ejecutando y entonces cada programa decide, según lo que se haya programado, qué hace para dar respuesta a esos eventos.

Cada vez que el usuario realiza una determinada acción sobre una aplicación que estamos programando en Java (un clic sobre el ratón, presionar una tecla, etc.), se produce un evento que el sistema operativo transmite a Java.

Java crea un objeto de una determinada clase de evento, y este evento se transmite a un determinado método para que lo gestione.

El **modelo de eventos de Java está basado en delegación**, es decir, la responsabilidad de gestionar un evento que ocurre en un objeto fuente la tiene otro objeto **oyente**.

Las **fuentes de eventos** (event sources) son objetos que detectan eventos y notifican a los receptores que se han producido dichos eventos. Ejemplos de fuentes:

- ✔ Botón sobre el que se pulsa o pincha con el ratón.
- ✔ Campo de texto que pierde el foco.
- ✔ Campo de texto sobre el que se presiona una tecla.
- ✔ Ventana que se cierra.
- ✔ etc.

En el apartado anterior de creación de interfaces con ayuda de los asistentes del **IDE**, vimos lo fácil que es realizar este tipo de programación, ya que el **IDE** hace muchas cosas, genera código automáticamente por nosotros.

Pero también podríamos hacerlo nosotros todo, si no tuviéramos un **IDE** como NetBeans, o porque simplemente nos apeteciera hacerlo todo desde código, sin usar asistentes ni diseñadores gráficos. En este caso, **los pasos a seguir** se pueden resumir en:

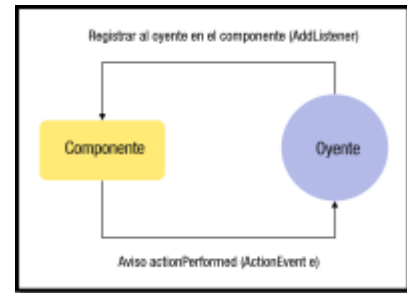
1. Crear la clase oyente que implemente la interfaz.
 - ✔ Ejemplo: ActionListener: pulsar un botón.
2. Implementar en la clase oyente los métodos de la interfaz.
 - ✔ Ejemplo: void actionPerformed(ActionEvent e).
3. Crear un objeto de la clase oyente y registrarlo como oyente en uno o más componentes gráficos que proporcionen interacción con el usuario.

Observa un ejemplo muy sencillo para ver estos tres pasos:

[Resumen textual alternativo](#)

Explicación del modelo de eventos con un ejemplo sencillo:

[Resumen textual alternativo](#)





Autoevaluación

Con la programación guiada por eventos, el programador se concentra en estar continuamente leyendo las entradas de teclado, de ratón, etc., para comprobar cada entrada o interacción producida por el usuario.

Verdadero

☐

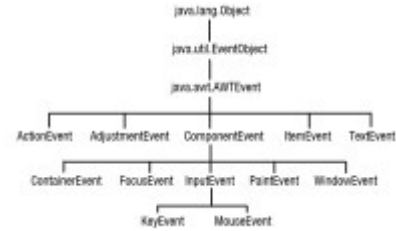
Falso

☐

4.3.- Tipos de eventos.

En la mayor parte de la literatura escrita sobre Java, encontrarás dos tipos básicos de eventos:

- ✓ **Físicos** o de **bajo nivel**: que corresponden a un evento hardware claramente identificable. Por ejemplo, se pulsó una tecla (*KeyStrokeEvent*). Destacar los siguientes:
 - ➡ En componentes: **ComponentEvent**. Indica que un componente se ha movido, cambiado de tamaño o de visibilidad
 - ➡ En contenedores: **ContainerEvent**. Indica que el contenido de un contenedor ha cambiado porque se añadió o eliminó un componente.
 - ➡ En ventanas: **WindowEvent**. Indica que una ventana ha cambiado su estado.
 - ➡ **FocusEvent**, indica que un componente ha obtenido o perdido la entrada del foco.
- ✓ **Semánticos** o de mayor nivel de abstracción: se componen de un conjunto de eventos físicos, que se suceden en un determinado orden y tienen un significado más abstracto. Por ejemplo: el usuario elige un elemento de una lista desplegable (*ItemEvent*).
 - ➡ **ActionEvent**, **ItemEvent**, **TextEvent**, **AdjustmentEvent**.



Los eventos en Java se organizan en una jerarquía de clases:

- ✓ La clase `java.util.EventObject` es la clase base de todos los eventos en Java.
- ✓ La clase `java.awt.AWTEvent` es la clase base de todos los eventos que se utilizan en la construcción de GUI.
- ✓ Cada tipo de evento *loqueseaEvent* tiene asociada una interfaz *loqueseaListener* que nos permite definir manejadores de eventos.
- ✓ Con la idea de simplificar la implementación de algunos manejadores de eventos, el paquete `java.awt.event` incluye clases `loqueseaAdapter` que implementan las interfaces *loqueseaListener*.



Autoevaluación

El evento que se dispara cuando le llega el foco a un botón es un evento de tipo físico.

Verdadero

☐

Falso

☐

4.4.- 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.



Long Zheng

KeyListener (oyente de teclas).

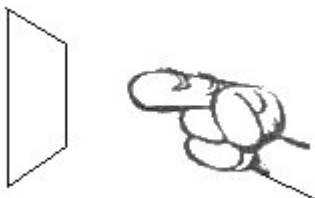
Método	Causa de la invocación
keyPressed (KeyEvent e)	Se ha pulsado una tecla.
keyReleased (KeyEvent e)	Se ha liberado una tecla.
keyTyped (KeyEvent e)	Se ha pulsado (y a veces soltado) una tecla.

KeyEvent (evento de teclas)

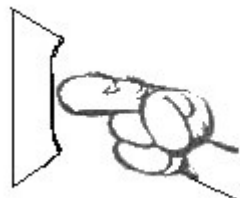
Métodos más usuales	Explicación
char getKeyChar()	Devuelve el carácter asociado con la tecla pulsada.
int getKeyCode()	Devuelve el valor entero que representa la tecla pulsada.
String getKeyText()	Devuelve un texto que representa el código de la tecla.
Object getSource()	Método perteneciente a la clase EventObject. Indica el objeto que produjo el evento.

La clase KeyEvent, define muchas constantes así:

- ✔ KeyEvent.VK_A especifica la tecla A.
- ✔ KeyEvent.VK_ESCAPE especifica la tecla ESCAPE.



Botón en estado normal.



Al pulsar la tecla se disparará el evento
KeyPressed.



Al liberar la tecla se genera el evento
KeyReleased.

En la siguiente presentación tienes el código del proyecto que te puedes descargar también a continuación. En él se puede ver un ejemplo del uso eventos. En concreto vemos cómo se están

capturando los eventos que se producen al pulsar una tecla y liberarla. El programa escribe en un área de texto las teclas que se oprimen.

[Resumen textual alternativo](#)

[Código Java comentado de un oyente de teclado](#) (20 KB)

4.5.- Eventos de ratón.

De manera similar 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.
mouseClicked (MouseEvent e)	Se ha pulsado y liberado un botón del ratón sobre un componente.
mouseEntered (<code>MouseEvent e)	Se ha entrado (con el puntero del ratón) en un componente.
mouseExited (<code>MouseEvent 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.

En el siguiente proyecto podemos ver una demostración de un formulario con dos botones. Implementamos un oyente `MouseListener` y registramos los dos botones para detectar tres de los cinco eventos del interface.

[Código Java comentado de un oyente de ratón](#) (28 KB)

Como se ve en el código, se deja en blanco el cuerpo de `mouseEntered` y de `mouseExited`, ya que no nos interesan en este ejemplo. Cuando se desea escuchar algún tipo de evento, de deben implementar todos los métodos del interface para que la clase no tenga que ser definida como abstracta. Para evitar tener que hacer esto, podemos utilizaradaptadores.



Gianfranco Degrandi

Debes conocer

Tienes un ejemplo de uso de los adaptadores en el siguiente enlace, en la página 143 de las 204 que tiene el documento:

[Ejemplo de Adaptadores](#) (1.96 MB)

Para saber más

En el enlace que ves a continuación, hay también un ejemplo interesante de la programación de eventos del ratón.

[La programación del ratón](#)



Autoevaluación

Cuando el usuario deja de pulsar una tecla se invoca a `keyReleased(KeyEvent e)`.

Verdadero

☐

Falso

☐

4.6.- Creación de controladores de eventos.

A partir del JDK1.4 se introdujo en Java la clase `EventHandler` para soportar oyentes de evento muy sencillos.

La utilidad de estos controladores o manejadores de evento es:

- ✓ Crear oyentes de evento sin tener que incluirlos en una clase propia.
- ✓ Esto aumente el rendimiento, ya que no "añade" otra clase.



Beverly & Pack

Como inconveniente, destaca la dificultad de construcción: **los errores no se detectan en tiempo de compilación**, sino en tiempo de ejecución.

Por esta razón, es mejor crear controladores de evento con la ayuda de un asistente y documentarlos todo lo posible.

El uso más sencillo de `EventHandler` consiste en instalar un oyente que llama a un método, en el objeto objetivo sin argumentos. En el siguiente ejemplo creamos un `ActionListener` que invoca al método `dibujar` en una instancia de `javax.swing.JFrame`.

```
miBoton.addActionListener(
    (ActionListener)EventHandler.create(ActionListener.class, frame, "dibujar");
```

Cuando se pulse `miBoton`, se ejecutará la sentencia `frame.dibujar()`. Se obtendría el mismo efecto, con mayor seguridad en tiempo de compilación, definiendo una nueva implementación al interface `ActionListener` y añadiendo una instancia de ello al botón:

```
// Código equivalente empleando una clase interna en lugar de EventHandler.
miBoton.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        frame.dibujar();
    }
});
```

Probablemente el uso más típico de `EventHandler` es extraer el valor de una propiedad de la fuente del objeto evento y establecer este valor como el valor de una propiedad del objeto destino. En el siguiente ejemplo se crea un `ActionListener` que establece la propiedad "label" del objeto destino al valor de la propiedad "text" de la fuente (el valor de la propiedad "source") del evento.

```
EventHandler.create(ActionListener.class, miBoton, "label", "source.text")
```

Esto correspondería a la implementación de la siguiente clase interna:

```
// Código equivalente utilizando una clase interna en vez de EventHandler.
new ActionListener {
```



```
public void actionPerformed(ActionEvent e) {  
    miBoton.setLabel(((JTextField)e.getSource()).getText());  
}  
}
```



Autoevaluación

El uso de EventHandler tiene como inconveniente, que los errores no se detectan en tiempo de ejecución.

Verdadero

☐

Falso

☐

Para saber más

Una de las novedades que trajo Java 8 fue las expresiones lambda. Con ellas, en algunos casos se puede escribir de una manera quizás más natural algunas expresiones, como por ejemplo la forma de añadir oyentes (listeners). Por ejemplo, en el siguiente enlace puedes ver para agregarle un oyente a un botón, dos formas de hacerlo, la usual y la forma con expresiones lambda:

[Ejemplos de uso de lambda](#)

[Más sobre lambda](#)

5.- Generación de programas en entorno gráfico.

Caso práctico



José Javier va de camino a la clase práctica en la que él, y el resto de la clase, van a probar a hacer sus primeros pasos en programación visual con entornos gráficos, el profesor les explica que al principio parece que todo es muy fácil sobre el papel, pero en realidad crear un proyecto con componentes gráficos no siempre es fácil. Pero el profesor lo tiene claro, hay que empezar por los contenedores que, como su propio nombre indica, se emplean para contener o ubicar al resto de componentes.

En este mismo tema, más arriba, has visto la lista de los principales componentes Swing que se incluyen en la mayoría de las aplicaciones, junto con una breve descripción de su uso, y una imagen que nos da una idea de cual es su aspecto.

También hemos visto un ejemplo de cómo crear con la ayuda de NetBeans unos sencillos programas. Ahora, vamos a ver con mayor detalle, los componentes más típicos utilizados en los programas en Java y cómo incluirlos dentro de una aplicación.



Manuel Martín

Citas para pensar

"Algo sólo es imposible hasta que alguien lo dude y termine probando lo contrario".

Albert Einstein

5.1.- Contenedores.

Por los ejemplos vistos hasta ahora, ya te habrás dado cuenta de la necesidad de que cada aplicación "contenga" de alguna forma esos componentes. ¿Qué componentes se usan para contener a los demás?

En Swing esa función la desempeñan un grupo de componentes llamados **contenedores** Swing.

Existen dos tipos de elementos contenedores:

- ✓ **Contenedores de alto nivel** o "peso pesado".
 - ➡ Marcos: **JFrame** y **JDialog** para aplicaciones
 - ➡ **JApplet**, para applets.
- ✓ **Contenedores de bajo nivel** o "peso ligero". Son los paneles: **JRootPane** y **JPanel**.

Cualquier aplicación, con interfaz gráfico de usuario típica, comienza con la apertura de una ventana principal, que suele contener la barra de título, los botones de minimizar, maximizar/restaurar y cerrar, y unos bordes que delimitan su tamaño.

Esa ventana constituye un marco dentro del cual se van colocando el resto de componentes que necesita el programador: menú, barras de herramientas, barra de estado, botones, casillas de verificación, cuadros de texto, etc.

Esa ventana principal o marco sería el contenedor de alto nivel de la aplicación.

Toda aplicación de interfaz gráfica de usuario Java tiene, al menos, un contenedor de alto nivel.

Los contenedores de alto nivel extienden directamente a una clase similar de AWT, es decir, **JFrame** extiende de **Frame**. Es decir, realmente necesitan crear una ventana del sistema operativo independiente para cada uno de ellos.

Los demás componentes de la aplicación no tienen su propia ventana del sistema operativo, sino que se dibujan en su objeto contenedor.

En los ejemplos anteriores del tema, hemos visto que podemos añadir un **JFrame** desde el diseñador de NetBeans, o bien escribiéndolo directamente por código. De igual forma para los componentes que añadamos sobre él.



Debes conocer

En el siguiente enlace se estudia cómo crear y abrir ventanas en Java, mediante **JFrame** para la ventana principal de la aplicación y **JDialog** para ventanas hijas. Fíjate que un **JDialog** impedirá el acceso a otras ventanas hasta que se cierre, si la ponemos como modal. Lo verás también más claro en el ejemplo resuelto del anexo II.

JFrame y JDialog

A continuación puedes ver cómo crear un **JInternalFrame**, que consisten en una ventana que va metida dentro de un panel y no puede salirse de él.

Para saber más

Puedes averiguar más sobre cuadros de diálogo en el enlace que tienes a continuación.

[Cuadros de diálogo modales y no modales en Java.](#)



Autoevaluación

Cualquier componente de gráfico de una aplicación Java necesita tener su propia ventana del sistema operativo.

Verdadero

☐

Falso

☐

Ejercicio Resuelto

Te presentamos una versión reducida de una tarea de años anteriores. Está reducida porque sólo nos interesa en este momento que te fijas en la clase principal. Concretamente, lo que pretendemos es que te fijas de qué manera podemos pasar información desde un JFrame a una ventana hija que se vaya a abrir, así como la manera de recuperar información desde esa nueva ventana abierta, antes de cerrarla y recuperar información hacia el formulario principal.

Como puedes ver en el código del proyecto, al ejecutar el programa y pinchar en la opción de crear sellos, lo que se va a ejecutar es el método `jMenuCrearSelloActionPerformed` que hay en la línea 177 de la clase Principal. Pues bien, en ese método observa que se declara la ventana que vamos a abrir y a continuación le mandamos la información que nos interesa en el constructor:

```
// Construcción de ventana secundaria.<br />dlgAlta = new EditaSel
```

en este caso la lista de objetos sello que tenemos en la clase principal, se la estamos pasando como tercer parámetro. También, como segundo parámetro le estamos pasando un booleano `true` para indicar que queremos que la ventana se abra como modal, es decir que mientras que no se cierre esa nueva ventana no pueda volver al formulario principal o abrir otras ventanas de la aplicación. Fíjate que el constructor de `EditaSello` tiene esta cabecera:

```
/**<br /> * Creates new form EditaSello<br /> * @param parent<br /
```

Observa también, que en en esta clase EditaSello, cuando le demos a aceptar, para volver al formulario padre, concretamente en la línea 319 de esa clase, lo que hacemos es ocultar el formulario, es decir, en ese momento está oculto pero no se ha borrado de la memoria, no se ha destruido todavía. Entonces, el programa se estará ejecutando en la línea 194 de la clase Principal, y es en esa línea donde estamos recogiendo la información de la ventana que se va a cerrar mediante ese método getSello(). Más adelante, una vez recogida la información destruiremos el formulario con dlgAlta.dispose();

5.2.- Cerrar la aplicación.

Cuando quieres terminar la ejecución de un programa, ¿qué sueles hacer? Pues normalmente pinchar en el icono de cierre de la ventana de la aplicación.

En Swing, una cosa es cerrar una ventana, y otra es que esa ventana deje de existir completamente, o cerrar la aplicación completamente.



francois schnell

- ✔ **Se puede hacer que una ventana no esté visible**, y sin embargo que ésta siga existiendo y ocupando memoria para todos sus componentes, usando el **método setVisible(false)**. En este caso bastaría ejecutar para el **JFrame** el método **setVisible(true)** para volver a ver la ventana con todos sus elementos.
- ✔ **Si queremos cerrar la aplicación**, es decir, que no sólo se destruya la ventana en la que se mostraba, sino que se destruyan y liberen todos los recursos (memoria y **CPU**) que esa aplicación tenía reservados, tenemos que invocar al método **System.exit(0)**.
- ✔ **También se puede invocar para la ventana JFrame al método dispose()**, heredado de la clase **Window**, que no requiere ningún argumento, **y que borra todos los recursos de pantalla usados por esta ventana y por sus componentes, así como cualquier otra ventana que se haya abierto como hija de esta** (dependiente de esta). Cualquier memoria que ocupara esta ventana y sus componentes se libera y se devuelve al sistema operativo, y tanto la ventana como sus componentes se marcan como "no representables". Y sin embargo, el objeto ventana sigue existiendo, y podría ser reconstruido invocando al método **pack()** o la método **show()**, aunque deberían construir de nuevo toda la ventana.

Las ventanas **JFrame** de Swing permiten establecer una operación de cierre por defecto con el método **setDefaultCloseOperation()**, definido en la clase **JFrame**.

¿Cómo se le indica al método el modo de cerrar la ventana?

Los valores que se le pueden pasar como parámetros a este método son una serie de constantes de clase:

- ✔ **DO_NOTHING_ON_CLOSE**: **no hace nada**, necesita que el programa maneje la operación en el método **windowClosing()** de un objeto **WindowListener** registrado para la ventana.
- ✔ **HIDE_ON_CLOSE**: **Oculto** de ser mostrado en la pantalla pero no destruye el marco o ventana después de invocar cualquier objeto **WindowListener** registrado.
- ✔ **DISPOSE_ON_CLOSE**: **Oculto y termina (destruye) automáticamente el marco o ventana** después de invocar cualquier objeto **WindowListener** registrado.
- ✔ **EXIT_ON_CLOSE**: Sale de la aplicación usando el método **System.exit(0)**. Al estar definida en **JFrame**, se puede usar con aplicaciones, pero no con applets.



Autoevaluación

System.exit(0) oculta la ventana pero no libera los recursos de CPU y memoria que la aplicación tiene reservados.

Verdadero

☐

Falso

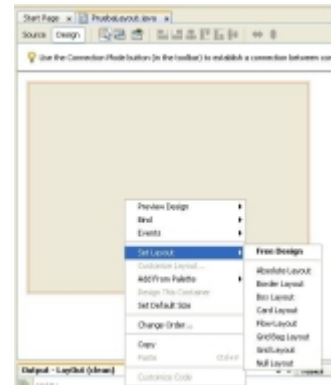
☐

5.3.- Organizadores de contenedores: layout managers.

Los layout managers son fundamentales en la creación de interfaces de usuario, ya que determinan las posiciones de los controles en un contenedor.

En lenguajes de programación para una única plataforma, el problema sería menor porque el aspecto sería fácilmente controlable. Sin embargo, dado que Java está orientado a la portabilidad del código, éste es uno de los aspectos más complejos de la creación de interfaces, ya que las medidas y posiciones dependen de la máquina en concreto.

En algunos entornos los componentes se colocan con coordenadas absolutas. En Java se desaconseja esa práctica porque en la mayoría de casos es imposible prever el tamaño de un componente.



Por tanto, en su lugar, se usan organizadores o también llamados **administradores de diseño** o **layout managers** o **gestores de distribución** que permiten colocar y maquetar de forma independiente de las coordenadas.

Debemos hacer un buen diseño de la interfaz gráfica, y así tenemos que elegir el mejor gestor de distribución para cada uno de los contenedores o paneles de nuestra ventana.

Esto podemos conseguirlo con el método `setLayout()`, al que se le pasa como argumento un objeto del tipo de Layout que se quiere establecer.

En NetBeans, una vez insertado un JFrame, si nos situamos sobre él y pulsamos botón derecho, se puede ver, como muestra la imagen, que aparece un menú, el cual nos permite elegir el layout que queramos.

Debes conocer

En la siguiente web puedes ver gráficamente los distintos layouts

[Layouts \(en inglés\)](#)

[Layouts con código comentado](#)

En la siguiente demostración se muestra el uso de FlowLayout para posicionar varios botones. Pulsa sobre cada opción para ver una imagen de su comportamiento con las propiedades anteriores:

[Resumen textual alternativo](#)



Autoevaluación

Cuando programamos en Java es aconsejable establecer coordenadas absolutas, siempre que sea posible, en nuestros componentes.

Verdadero

☐

Falso

☐

5.4.- Contenedor ligero: JPanel.

Es la clase utilizada como contenedor genérico para agrupar componentes.

Normalmente cuando una ventana de una aplicación con interfaz gráfica cualquiera presenta varios componentes, para hacerla más atractiva y ordenada al usuario se suele hacer lo siguiente:

- ✓ Crear un marco, un **JFrame**.
- ✓ Para organizar mejor el espacio en la ventana, añadiremos varios paneles, de tipo **JPanel**. (Uno para introducir los datos de entrada, otro para mostrar los resultados y un tercero como zona de notificación de errores.)
- ✓ Cada uno de esos paneles estará delimitado por un borde que incluirá un título. Para ello se usan las clases disponibles en **BorderFactory** (**BevelBorder**, **CompoundBorder**, **EmptyBorder**, **EtchedBorder**, **LineBorder**, **LoweredBevelBorder**, **MatteBorder** y **TitledBorder**) que nos da un surtido más o menos amplio de tipos de bordes a elegir.
- ✓ En cada uno de esos paneles se incluyen las etiquetas y cuadros de texto que se necesitan para mostrar la información adecuadamente.



Con NetBeans es tan fácil como arrastrar tantos controles JPanel de la paleta hasta el JFrame. Por código, también es muy sencillo, por ejemplo podríamos crear un panel rojo, darle sus características y añadirlo al JFrame del siguiente modo:

```
...
JPanel panelRojo = new JPanel();
panelRojo.setBackground(Color.RED);
panelRojo.setSize(300,300);
// Se crea una ventana
JFrame ventana=new JFrame("Prueba en rojo");
ventana.setLocation(100,100);
ventana.setVisible(true);
// Se coloca el JPanel en el content pane
Container contentPane=ventana.getContentPane();
contentPane.add(panelRojo);
...
```

Cuando en Sun desarrollaron Java, los diseñadores de Swing, por alguna circunstancia, determinaron que para algunas funciones, como añadir un JComponent, los programadores no pudiéramos usar JFrame.add, sino que en lugar de ello, primeramente, tuviéramos que obtener el objeto Container asociado con JFrame.getContentPane(), y añadirlo.

Sun se dio cuenta del error y ahora permite utilizar JFrame.add, desde Java 1.5 en adelante. Sin embargo, podemos tener el problema de que un código desarrollado y ejecutado correctamente en 1.5 falle en máquinas que tengan instalado 1.4 o anteriores.

Para saber más

En la siguiente web puedes ver algo más sobre paneles.

[Paneles.](#)

5.5.- Etiquetas y campos de texto.

Los **cuadros de texto** Swing vienen implementados en Java por la clase **JTextField**.

Para insertar un campo de texto, el procedimiento es tan fácil como: **seleccionar el botón correspondiente a JTextField en la paleta de componentes**, en el diseñador, y **pinchar sobre el área de diseño** encima del panel en el que queremos situar ese campo de texto. El tamaño y el lugar en el que se sitúe, dependerá del Layout elegido para ese panel.



El componente Swing etiqueta **JLabel**, se utiliza para crear etiquetas de modo que podamos insertarlas en un marco o un panel para visualizar un texto estático, que no puede editar el usuario.

Los constructores son:

- ✓ JLabel(). Crea un objeto JLabel sin nombre y sin ninguna imagen asociada.
- ✓ JLabel(Icon imagen). Crea un objeto sin nombre con una imagen asociada.
- ✓ JLabel(Icon imagen, int alineacionHorizontal). Crea una etiqueta con la imagen especificada y la centra en horizontal.
- ✓ JLabel(String texto). Crea una etiqueta con el texto especificado.
- ✓ JLabel(String texto, Icon icono, int alineacionHorizontal). Crea una etiqueta con el texto y la imagen especificada y alineada horizontalmente.
- ✓ JLabel(String texto, int alineacionHorizontal). Crea una etiqueta con el texto especificado y alineado horizontalmente.

Si quieres ver las principales propiedades asociadas a **JTextField**, y los métodos que permiten modificarlas, puedes utilizar la presentación siguiente:

[Resumen textual alternativo](#)

Para saber más

En el siguiente enlace puedes ver cómo usar DecimalFormat para presentar un número en un JTextField o recoger el texto del JTextField y reconstruir el número.

[Formatear cuadro de texto.](#)



Autoevaluación

Un componente JLabel permite al usuario de la aplicación en ejecución cambiar el texto que muestra dicho componente.

Verdadero

☐

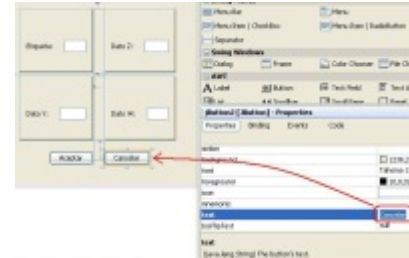
Falso

☐

5.6.- Botones.

Ya has podido comprobar que prácticamente todas las aplicaciones incluyen botones que al ser pulsados efectúan alguna acción: hacer un cálculo, dar de alta un libro, aceptar modificaciones, etc.

Estos botones se denominan **botones de acción**, precisamente porque realizan una acción cuando se pulsan. En Swing, la clase que los implementa en Java es la JButton.



Los principales métodos son:

- ✔ void setText(String). Asigna el texto al botón.
- ✔ String getText(). Recoge el texto.

Se pueden asociar imágenes con los botones, mediante ImageIcon, que permite especificar fácilmente un archivo de imagen (jpeg o GIF, incluyendo GIF animados). La forma más fácil de asociar una imagen con un JButton es pasar el ImageIcon al constructor. Un JButton realmente permite siete imágenes asociadas:

1. La imagen principal (se usa setIcon para especificarla, si no se proporciona en el constructor).
2. La imagen a usar cuando el botón se presiona (setPressedIcon).
3. La imagen a usar cuando el ratón está sobre el (setRolloverIcon, aunque tienes que llamar primero a setRolloverEnabled(true)).
4. La imagen a usar cuando el botón está deshabilitado (setDisabledIcon),
5. La imagen a usar cuando el botón está seleccionado y habilitado (setSelectedIcon).
6. La imagen a usar cuando está seleccionado pero deshabilitado (setDisabledSelectedIcon).
7. La imagen a utilizar cuando el ratón está sobre ello mientras se selecciona (setRolloverSelectedIcon).

A continuación puedes descargar un ejemplo de cuatro botones similares: uno con texto plano, uno con sólo una imagen, uno con ambos y el diseño por defecto (imagen a la izquierda, texto a la derecha) y uno con las posiciones de imagen y texto invertidas.

[Diseño de una GUI sencilla.](#) (0.70 MB)

Hay un tipo especial de botones, que se comportan como **interruptores de dos posiciones** o estados (pulsados-on, no pulsados-off). Esos botones especiales se denominan botones on/off o JToggleButton.

Para saber más

A continuación puedes ver un enlace en el que se diseña una interfaz gráfica de usuario sencilla, con los controles que hemos visto hasta ahora.

[Diseño de una GUI sencilla.](#) (0.70 MB)

5.6.1.- Ejemplo de gestión de eventos para un botón: ActionListener.

El trabajo con **interfaces** es algo habitual en el desarrollo de aplicaciones en Java. Es por tanto muy importante comprender correctamente su funcionamiento y la interacción con las distintas **bibliotecas (paquetes de clases e interfaces)** que proporcionan las APIs. Estas **clases** e **interfaces** son fundamentales para la creación de aplicaciones y tendrás que utilizarlas en multitud de ocasiones (además, por supuesto, de las que tengas que desarrollar por ti mismo).



Vamos a ver un ejemplo de una **interfaz** proporcionada por la API de Java que puede ser implementada por alguna clase creada por ti dentro de una pequeña aplicación. Hemos escogido la interfaz ActionListener.

La **interfaz** ActionListener ya la has utilizado en la unidad dedicada a las **interfaces gráficas**. Las clases que quieran realizar una determinada acción cada vez que se produzca cierto **evento** en el sistema deben implementar esta **interfaz**. Este tipo de **interfaces** se encuentran dentro de los **Event Listeners** u "oyentes de eventos" y son útiles para detectar que se ha producido un determinado **evento** asíncrono durante la ejecución de tu aplicación (pulsación de una tecla, clic de ratón, etc.). Son intensivamente utilizadas en el desarrollo de las **interfaces gráficas de usuario**.

Para saber más

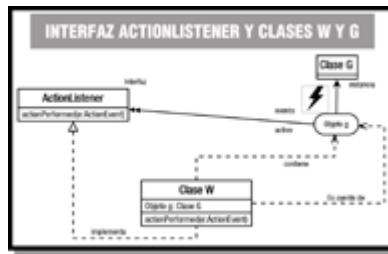
Si deseas profundizar algo más en el tema de los Event Listeners puedes echar un vistazo a los tutoriales de iniciación de Java en el sitio web de Oracle (en inglés) en el que se explican detalladamente ejemplos sencillos así como otros bastante más complejos:

[Introduction to Event Listeners.](#)

Vamos a ver un ejemplo lo más sencillo posible: se trata de desarrollar una pequeña aplicación de escritorio (utilizando la API de **Swing**) con una **ventana** que contenga un par de **botones** y que al pulsar alguno de esos **botones (evento)**, la aplicación sea capaz de detectar que se ha producido ese **evento** y por tanto realizar una determinada **acción** (por ejemplo generar un sonido o mostrar un determinado texto).

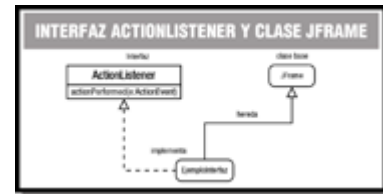
Para ello es necesario que la clase que vaya a gestionar el evento sea un "**oyente**" de ese evento, es decir, que sea capaz de enterarse de que se ha producido ese evento. Esa capacidad o habilidad la proporciona la API de Java a través de las interfaces de tipo "**oyente**". En concreto vamos a implementar la **interfaz ActionListener**.

La interface ActionListener contiene un único método: actionPerformed. Toda clase **W** que implemente la esta interfaz tendrá que definir ese método. Si incrustamos un determinado objeto gráfico **G** en una ventana **W** y asociamos a ese objeto gráfico la clase **W**, indicando que es su **oyente y gestor de eventos**, cuando se produzca algún evento sobre **G** se ejecutará el método actionPerformed implementado por **W**.



En nuestro ejemplo podríamos hacer lo siguiente:

- ✓ Definir una **ventana principal**, por ejemplo una clase basada en el tipo **marco** o **frame** (subclase de JFrame) que implemente la **interfaz** ActionListener.
- ✓ Definir uno o varios **botones** (objetos de la clase JButton) dentro de la ventana.
- ✓ Asociar como **oyente** a los objetos de tipo **botón** la propia **ventana principal** (frame), pues va a ser capaz de gestionar **eventos** (implementa el método actionPerformed).
- ✓ Implementar el método actionPerformed como un **método de la ventana principal** (frame) para **reaccionar** de algún modo cuando se produzca algún **evento desencadenado por un botón** (pues se ha establecido a la ventana principal como **oyente** y **gestora de los eventos** del botón).



En la siguiente presentación puedes observar detalladamente el proceso completo para llevar a cabo este ejemplo de implementación de la interfaz ActionListener:

[Resumen textual alternativo](#)

Desde el siguiente enlace puedes descargar el ejemplo completo en el que se implementa la interfaz ActionListener en un pequeño programa que crea una ventana que gestiona dos botones: uno de ellos produce un pitido y el otro produce dos pitidos.

[Proyecto EjemploInterfazActionListener](#). (22 KB)

5.7.- Casillas de verificación y botones de radio.

Las casillas de verificación en Swing están implementadas para Java por la clase **JCheckBox**, y los botones de radio o de opción por la clase **JRadioButton**. Los grupos de botones, por la clase **ButtonGroup**.

La funcionalidad de ambos componentes es en realidad la misma.

- ✓ Ambos tienen **dos "estados"**: seleccionados o no seleccionados (marcados o no marcados).
- ✓ Ambos **se marcan o desmarcan** usando el método **setSelected(boolean estado)**, que establece el valor para su propiedad **selected**. (El estado toma el valor **true** para seleccionado y **false** para no seleccionado).
- ✓ A ambos le **podemos preguntar si están seleccionados o no**, mediante el método **isSelected()**, que devuelve **true** si el componente está seleccionado y **false** si no lo está.
- ✓ Para ambos **podemos asociar un icono distinto** para el estado de **seleccionado** y el de **no seleccionado**.



JCheckBox pueden usarse en menús mediante la clase **JCheckBoxMenuItem**.

JButtonGroup permite agrupar una serie de casillas de verificación (**JRadioButton**), de entre las que sólo puede seleccionarse una. Marcar una de las casillas implica que el resto sean desmarcadas automáticamente. La forma de hacerlo consiste en añadir un **JButtonGroup** y luego, agregarle los botones.

Cuando en **un contenedor** aparezcan **agrupados** varios **botones de radio** (o de opción), **entenderemos** que no son opciones independientes, sino que sólo uno de ellos podrá estar activo en cada momento, y necesariamente uno debe estar activo. Por tanto en ese contexto entendemos que son opciones excluyentes entre sí.

Para saber más

Breve enlace donde se aprecia el uso de casillas de verificación.

[Casillas de verificación.](#)

En el siguiente enlace puedes ver un videotutorial para crear un ejemplo básico de Java con interfaz gráfica.

[Resumen textual alternativo](#)



Autoevaluación

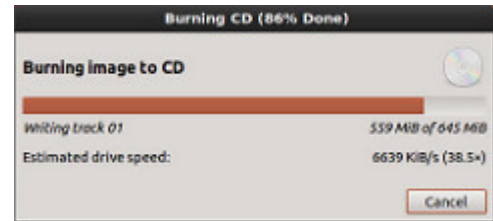
Los componentes radiobotones y las casillas de verificación tienen ambos dos

estados: seleccionado y no seleccionado.

Verdadero ☐ Falso ☐

5.8.- Barras de progreso.

¿Alguna vez has creído que un programa se te había quedado "colgado" y no respondía, y realmente sí estaba pasando algo, y finalmente terminaba el proceso? Seguro que más de una vez, y la solución para que no ocurra eso es poner una barra de progreso que muestre al usuario que el proceso está ahí, que algo está pasando aunque sea lentamente.



[John Baer.](#)

En la web encontrarás un montón de ejemplos de implementación, alguno con hilos, a continuación, en el siguiente enlace te mostramos un enlace a una implementación muy sencilla y explicada.

Debes conocer

Interesante ejemplo explicado sobre las barras de progreso en Java.

[Barra de progreso.](#)

Para saber más

Otro ejemplo de barra de progreso, implementado con hilos.

[Barra de progreso con hilos.](#)

5.9.- Listas.

En casi todos los programas, nos encontramos con que se pide al usuario que introduzca un dato, y además un dato de entre una lista de valores posibles, no un dato cualquiera.

La clase JList constituye un componente lista sobre el que se puede ver y seleccionar uno o varios elementos a la vez. En caso de haber más elementos de los que se pueden visualizar, es posible utilizar un componente JScrollPane para que aparezcan barras de desplazamiento.



Podemos ver cómo añadir un JList en NetBeans en la siguiente presentación:

[Resumen textual alternativo](#)

En los componentes JList, un modelo `ListModel` representa **los contenidos de la lista**. Esto significa que los datos de la lista se guardan en una estructura de datos independiente, denominada modelo de la lista. Es fácil mostrar en una lista los elementos de un vector o array de objetos, usando un constructor de JList que cree una instancia de ListModel automáticamente a partir de ese vector.

Vemos a continuación un ejemplo para crear una lista JList que muestra las cadenas contenidas en el vector `info[]`:

```
String[] info = {"Pato", "Loro", "Perro", "Cuervo"};
JList listaDatos = new JList(info);
/* El valor de la propiedad model de JList es un objeto que proporciona una
El método getModel() permite recoger ese modelo en forma de Vector de objetos
Vector, como getElementAt(i), que proporciona el elemento de la posición i
for (int i = 0; i < listaDatos.getModel().getSize(); i++) {
System.out.println(listaDatos.getModel().getElementAt(i));
}
```

Para saber más

A continuación puedes ver como crear un JList, paso a paso, en el enlace siguiente.

[Crear un JList paso a paso.](#)

En el enlace que tienes a continuación puedes ver una presentación que versa sobre listas en Swing.

[Resumen textual alternativo](#)

5.9.1.- Listas (II).

Cuando trabajamos con un componente JList, podemos seleccionar un único elemento, o varios elementos a la vez, que a su vez pueden estar contiguos o no contiguos. La posibilidad de hacerlo de una u otra manera la establecemos con la propiedad `selectionMode`.



Los valores posibles para la propiedad `selectionMode` para cada una de esas opciones son las siguientes constantes de clase del interface **ListSelectionModel**:

- ✔ **MULTIPLE_INTERVAL_SELECTION**: Es el valor por defecto. Permite seleccionar múltiples intervalos, manteniendo pulsada la tecla `CTRL`, mientras se seleccionan con el ratón uno a uno, o la tecla de mayúsculas, mientras se pulsa el primer elemento y el último de un intervalo.
- ✔ **SINGLE_INTERVAL_SELECTION**: Permite seleccionar un único intervalo, manteniendo pulsada la tecla mayúsculas mientras se selecciona el primer y último elemento del intervalo.
- ✔ **SINGLE_SELECTION**: Permite seleccionar cada vez un único elemento de la lista.

Podemos establecer el valor de la propiedad `selectedIndex` con el método `setSelectedIndex()` es decir, el índice del elemento seleccionado, para seleccionar el elemento del índice que se le pasa como argumento.

Como hemos comentado, los datos se almacenan en un modelo que al fin y al cabo es un vector, por lo que tiene sentido hablar de índice seleccionado.

También se dispone de un método `getSelectedIndex()` con el que podemos averiguar el índice del elemento seleccionado.

El método `getSelectedValue()` devuelve el objeto seleccionado, de tipo `Object`, sobre el que tendremos que aplicar un "casting" explícito para obtener el elemento que realmente contiene la lista (por ejemplo un `String`). Observa que la potencia de usar como modelo un vector de `Object`, es que en el `JList` podemos mostrar realmente cualquier cosa, como por ejemplo, una imagen.

El método `setSelectedValue()` permite establecer cuál es el elemento que va a estar seleccionado.

Si se permite hacer selecciones múltiples, contamos con los métodos:

- ✔ `setSelectedIndices()`, al que se le pasa como argumento un vector de enteros que representa los índices a seleccionar.
- ✔ `getSelectedIndices()`, que devuelve un vector de enteros que representa los índices de los elementos o ítems que en ese momento están seleccionados en el `JList`.
- ✔ `getSelectedValues()`, que devuelve un vector de `Object` con los elementos seleccionados en ese momento en el `JList`.

Debes conocer

En el siguiente enlace tienes algún ejemplo comentado para crear JList

[JList](#) (2.79 MB)



Autoevaluación

Los componentes JList permiten la selección de elementos de una lista, siempre que estén uno a continuación del otro de manera secuencial.

Verdadero

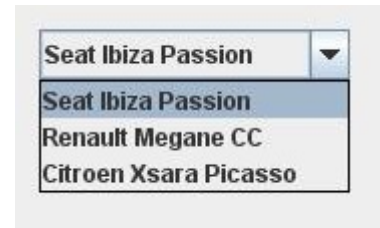
☐

Falso

☐

5.10.- Listas desplegables.

Una **lista desplegable** se representa en Java con el componente Swing **JComboBox**. Consiste en una lista en la que sólo se puede elegir una opción. Se pueden crear **JComboBox** tanto editables como no editables.



Una lista desplegable es una mezcla entre un campo de texto editable y una lista. Si la propiedad **editable** de la lista desplegable la fijamos a verdadero, o sea a **true**, el usuario podrá seleccionar uno de los valores de la lista que se despliega al pulsar el botón de la flecha hacia abajo y dispondrá de la posibilidad de teclear directamente un valor en el campo de texto.

Establecemos la propiedad **editable** del **JComboBox** el método **setEditable()** y se comprueba con el método **isEditable()**. La clase **JComboBox** ofrece una serie de métodos que tienen nombres y funcionalidades similares a los de la clase **JList**.

Debes conocer

En el siguiente vídeo se muestra cómo añadir un JComboBox en NetBeans.

Uso de JComboBox

[Resumen textual alternativo](#)

Puedes ver en el siguiente videotutorial cómo se crea una aplicación con NetBeans, en la que se van añadiendo los controles que hemos visto, entre ellos una lista desplegable.

[Resumen textual alternativo](#)

Para saber más

En el siguiente enlace tienes algún ejemplo comentado para crear JComboBox por código.

[Ejemplo JComboBox](#)

5.11.- Menús.

En las aplicaciones informáticas siempre se intenta que el usuario disponga de un menú para facilitar la localización una operación. La filosofía, al crear un menú, es que contenga todas las acciones que el usuario pueda realizar en la aplicación. Lo más normal y útil es hacerlo clasificando o agrupando las operaciones por categorías en submenús.

En Java usamos los componentes `JMenu` y `JMenuItem` para crear un menú e insertarlo en una barra de menús. La barra de menús es un componente `JMenuBar`.

Podemos crear un menú con la paleta de componentes, la manera más fácil, o bien por código.



En la siguiente presentación se ve resumidamente cómo crear un menú con la paleta de NetBeans.

[Resumen textual alternativo](#)

También podríamos construir un menú directamente por código. Los constructores son los siguientes:

- ✓ `JMenu()`. Construye un menú sin título.
- ✓ `JMenu(String s)`. Construye un menú con título indicado por `s`.
- ✓ `JMenuItem()`. Construye una opción sin icono y sin texto.
- ✓ `JMenuItem(Icon icono)`. Construye una opción con icono y con texto.

Podemos construir por código un menú sencillo como el de la imagen con las siguientes sentencias:

```
// Crear la barra de menú
JMenuBar barra = new JMenuBar();
// Crear el menú Archivo
JMenu menu = new JMenu("Archivo");
// Crear las opciones del menú
JMenuItem opcionAbrir = new JMenuItem("Abrir");
menu.add(opcionAbrir);
JMenuItem opcionguardar = new JMenuItem("Guardar");
menu.add(opcionguardar);
JMenuItem opcionSalir = new JMenuItem("Salir");
menu.add(opcionSalir);
// Añadir las opciones a la barra
barra.add(menu);
// Establecer la barra
setJMenuBar(barra);
```

Frecuentemente, dispondremos de alguna opción dentro de un menú, que al elegirla, nos dará paso a un conjunto más amplio de opciones posibles.

Cuando en un menú un elemento del mismo es a su vez un menú, se indica con una flechita al final de esa opción, de forma que se sepa que, al seleccionarla, nos abrirá un nuevo menú.

Para incluir un menú como submenú de otro basta con incluir como ítem del menú, un objeto que

también sea un menú, es decir, incluir dentro del JMenu un elemento de tipo JMenu.



Autoevaluación

Un menú se crea utilizando el componente JMenu dentro de un JList.

Verdadero

☐

Falso

☐

5.11.1.- Separadores.

A veces, en un menú pueden aparecer distintas opciones. Por ello, nos puede interesar destacar un determinado grupo de opciones o elementos del menú como relacionados entre sí por referirse a un mismo tema, o simplemente para separarlos de otros que no tienen ninguna relación con ellos.

El **componente Swing que tenemos en Java para esta funcionalidad es**: `JSeparator`, que dibuja una línea horizontal en el menú, y así separa visualmente en dos partes a los componentes de ese menú. En la imagen podemos ver cómo se han separado las opciones de Abrir y Guardar de la opción de Salir, mediante este componente.



Si lo hacemos de manera gráfica, simplemente arrastramos un `JSeparator` a donde nos interese. Si lo hacemos por código, al código anterior, tan sólo habría que añadirle una línea, la que resaltamos en negrita:

```
...  
menu.add(opcionguardar);  
menu.add(new JSeparator());  
JMenuItem opcionSalir = new JMenuItem("Salir");  
...
```



Autoevaluación

En un menú en Java se debe introducir siempre un separador para que se pueda compilar el código.

Verdadero ☐ Falso ☐

Para saber más

En este enlace podrás encontrar más información sobre diseño de menús.

[Diseño de menús.](#) (3.75 MB)

5.11.2.- Aceleradores de teclado y mnemónicos.

A veces, tenemos que usar una aplicación con interfaz gráfica y no disponemos de ratón, porque se nos ha roto, o cualquier causa.

Además, cuando diseñamos una aplicación, debemos preocuparnos de las características de accesibilidad.



Algunas empresas de desarrollo de software obligan a todos sus empleados a trabajar sin ratón al menos un día al año, para obligarles a tomar conciencia de la importancia de que todas sus aplicaciones deben ser accesibles, usables sin disponer de ratón.

Ya no sólo en menús, sino en cualquier componente interactivo de una aplicación: campo de texto, botón de acción, lista desplegable, etc., es muy recomendable que pueda seleccionarse sin el ratón, haciendo uso exclusivamente del teclado.

Para conseguir que nuestros menús sean accesibles mediante el teclado, la idea es usar aceleradores de teclado o **atajos de teclado** y de **mnemónicos**.

Un acelerador o **atajo de teclado** es una combinación de teclas que se asocia a una opción del menú, de forma que pulsándola se consigue el mismo efecto que abriendo el menú y seleccionando esa opción.

Esa combinación de teclas **aparece escrita a la derecha de la opción del menú**, de forma que el usuario pueda tener conocimiento de su existencia.

Para añadir un atajo de teclado, lo que se hace es emplear la propiedad **accelerator** en el diseñador

Los atajos de teclado **no suelen** asignarse a todas las opciones del menú, sino **sólo a** las que más se usan.

Un **mnemónico** consiste en resaltar una letra dentro de esa opción, mediante un subrayado, de forma que pulsando **Alt +** se abra el menú correspondiente. Por ejemplo en la imagen con **Alt + A** abríamos ese menú que se ve, y ahora con **Ctrl+G** se guardaría el documento.

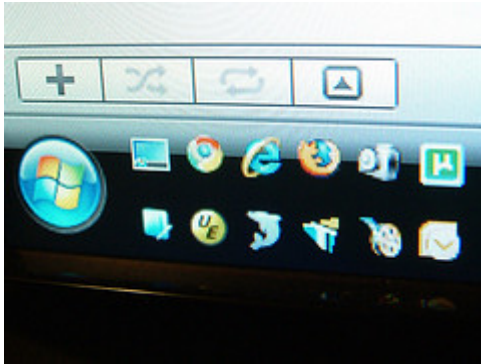
Para añadir mediante código un mnemónico a una opción del menú, se hace mediante la **propiedad mnemonic**. Los mnemónicos sí que deberían ser incluidos para todas las opciones del menú, de forma que todas puedan ser elegidas haciendo uso sólo del teclado, mejorando así la accesibilidad de nuestra aplicación. Para ver cómo se añadiría mediante el diseñador de NetBeans, mira el siguiente Debes conocer.

Debes conocer

En el siguiente enlace se puede ver cómo añadir a una aplicación que estemos construyendo con NetBeans, entre otras cosas, mnemónicos y aceleradores.

[Diseñando con NetBeans](#)

5.12.- Barras de herramientas.



JAK SIE MASZ

Frecuentemente, en la mayoría de aplicaciones informáticas se introducen Barras de Botones con imágenes de las operaciones principales (o más habituales) que realiza dicha aplicación. Especialmente para abrir y cerrar ficheros, guardar cambios, o las típicas cortar, copiar y pegar, etc.

¿Crees que será útil disponer de la misma funcionalidad repetida en varias partes de la aplicación?

Por **ejemplo**, si ya tenemos la opción Salir en un menú para terminar la ejecución de la aplicación, ¿resultará útil disponer de esa funcionalidad en algún otro sitio, como

por ejemplo un botón?

Desde luego, resulta tan útil, que es lo que suelen hacer la mayoría de las aplicaciones, repetir la funcionalidad de sus menús en botones.

Normalmente será rentable repetir funcionalidades para aquellos aspectos de nuestra aplicación que se ejecuten con mucha frecuencia, garantizando un acceso más rápido que por medio de los menús. En tales casos, a estas opciones se les suele asociar un icono, de forma que el botón que duplica la funcionalidad lo único que contiene es ese icono, usualmente sin texto alguno.

Por tanto es el icono común el que identifica que se trata de una función común.

Pero imagina que tenemos muchas funciones en nuestra aplicación que por su importancia queremos tener repetidas en botones. Podría resultar incómodo ir añadiendo montones de botones sueltos por toda la ventana de nuestra aplicación. Por tanto, resulta mucho más cómodo agruparlos todos en una barra de herramientas. Seguramente te resulta familiar el uso de estas barras de herramientas, ya que la mayoría de las aplicaciones disponen de alguna de ellas, pero ¿cómo se añade una barra de herramientas a nuestra aplicación Java?

- **En Java las barras de herramientas se añaden como objetos de la clase `JToolBar`, y haciendo uso del diseñador.**
- **Basta con añadir un objeto `JToolBar` con el ratón al área de diseño, y luego añadir los botones a esa barra de tareas, en vez de colocarlos en un panel, por ejemplo.**
- Si queremos que los botones sólo contengan una imagen, basta con eliminarles el texto en la **propiedad text** y asociarles un icono con la **propiedad icon** desde el diseñador.

Las principales propiedades de una barra de herramientas **`JToolBar`** son:

- **floatable:** Indica que va a ser (o no) una barra flotante, que va a poder desplazarse en forma de ventana flotante al lugar de la ventana de la aplicación al que la desplazemos arrastrándola con el ratón, o estar fija en una posición.
- **orientation:** Si toma el valor 0, la barra estará orientada en horizontal. Si toma el valor 1, estará orientada en vertical.
- **rollover:** Indica si los botones de la barra van a tener (o no) bordes alrededor

Por último añadir que es conveniente que todos los botones de una barra de herramientas tengan las mismas dimensiones, para que el aspecto sea visualmente más agradable, (requisito que no cumplen los de la imagen de ejemplo).

Resumen textual alternativo

Anexo I.- Principales clases AWT.

Principales clases AWT

NOMBRE DE LA CLASE AWT	UTILIDAD DEL COMPONENTE
Applet	Ventana para una applet que se incluye en una página web.
Button	Crea un botón de acción.
Canvas	Crea un área de trabajo en la que se puede dibujar. Es el único componente AWT que no tiene un equivalente Swing.
Checkbox	Crea una casilla de verificación.
Label	Crea una etiqueta.
Menu	Crea un menú.
ComboBox	Crea una lista desplegable.
List	Crea un cuadro de lista.
Frame	Crea un marco para las ventanas de aplicación.
Dialog	Crea un cuadro de diálogo.
Panel	Crea un área de trabajo que puede contener otros controles o componentes.
PopupMenu	Crea un menú emergente.
RadioButton	Crea un botón de radio.
ScrollBar	Crea una barra de desplazamiento.
ScrollPane	Crea un cuadro de desplazamiento.
TextArea	Crea un área de texto de dos dimensiones.
TextField	Crea un cuadro de texto de una dimensión.
Window	Crea una ventana.

Anexo II.- Ejercicio resuelto.

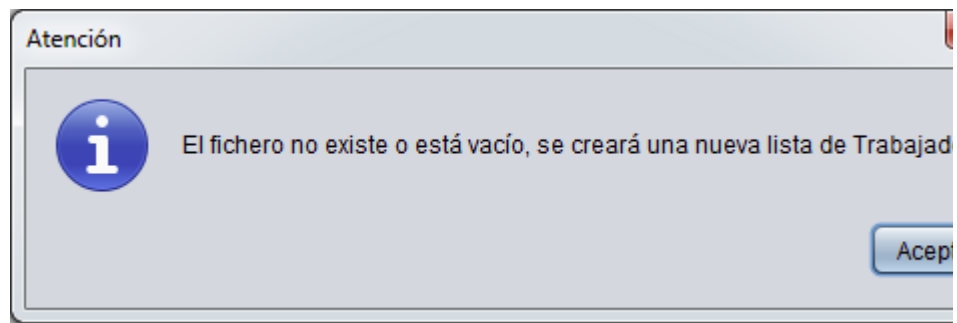
Tarea de cursos anteriores resuelta.

ENUNCIADO DE LA TAREA.

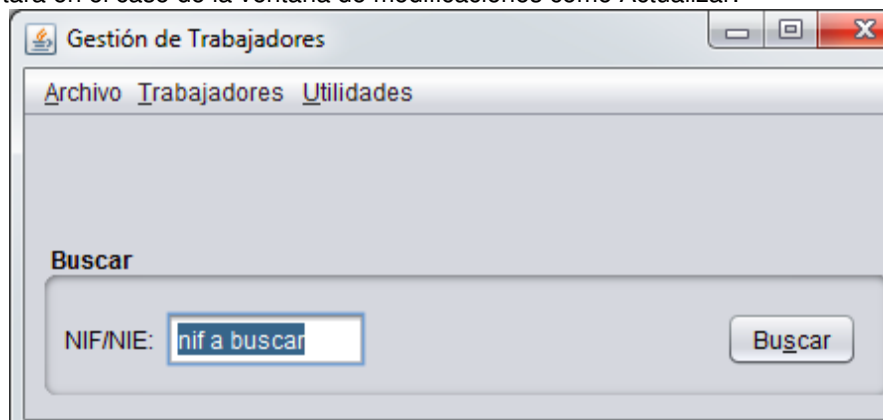
Actividad 1.- Dotar de una interfaz gráfica a la aplicación de gestión de trabajadores que venimos en las tareas anteriores.

¿Qué debe incluir nuestra aplicación?

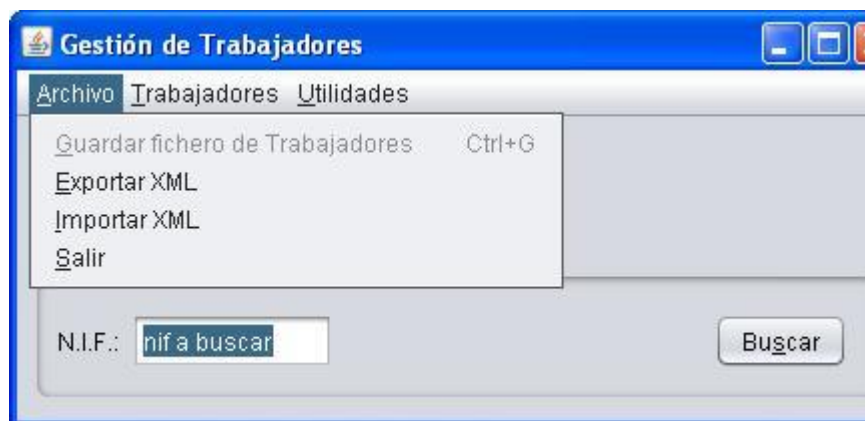
- ✓ Lo primero que debe hacer al ejecutar la aplicación es intentar cargar desde fichero (por ejemplo o donde los hayas guardado en la tarea anterior) al ArrayList donde se almacenarán los datos. Si no existe el fichero o está vacío, lógicamente el ArrayList estará vacío, sin ningún trabajador. En dicho fichero, se avisará al usuario de que la aplicación no tiene datos grabados, mediante un mensaje.



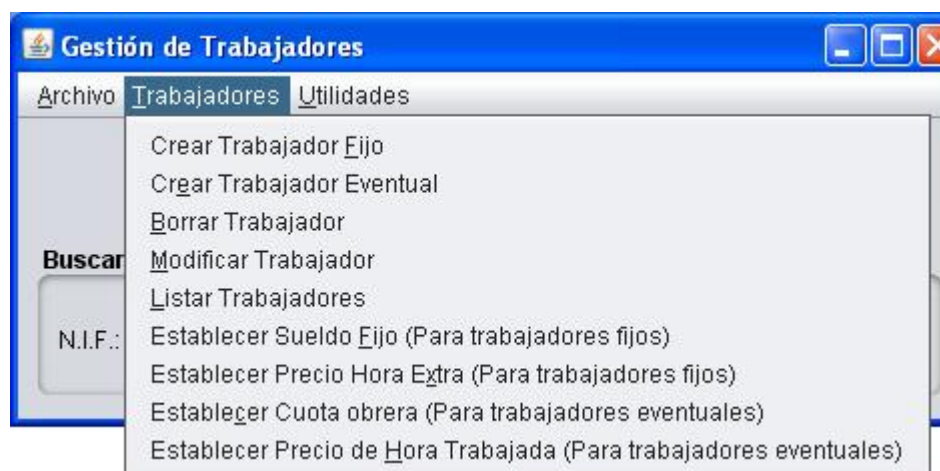
- ✓ Si existe el fichero, se cargarán los datos guardados de modo que los trabajadores irán al ArrayList y recuperarán los datos adicionales que haya (variables estáticas que deban recuperarse para la ejecución de la aplicación, y que continúe tal y como estaba cuando se guardaron los datos).
- ✓ Habrá una opción en el menú para guardar los datos en el fichero trabajadores.dat, que lógicamente los datos que comentábamos en el punto anterior.
- ✓ En caso de que al ejecutar la aplicación, se modifique algún dato, y el usuario intente salir de la aplicación sin guardar los datos, se le avisará indicando que si se sale sin guardar los datos, perderá los cambios. Podría ser con un mensaje como: **"Ha realizado cambios que no ha guardado en disco. ¿Debes guardar los cambios?(S/N)"**.
- ✓ A continuación se mostrará una pantalla similar a la siguiente, donde puedes ver que hay un menú superior y un panel más abajo, con un campo de texto donde se puede introducir un NIF para registrar un trabajador por su NIF o NIE. Al pulsar el botón Buscar, si el trabajador se encuentra en la lista, se mostrará una pantalla similar a la de Altas de trabajadores que se muestra en la Actividad 2, solo que en el título indicará que se trata de **Modificar Trabajador ...** indicando el tipo de que se trate (Fijo o Eventual). Guardar se etiquetará en el caso de la ventana de modificaciones como Actualizar:



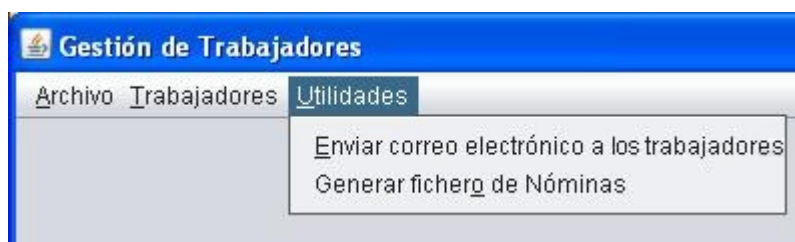
- ✓ En el menú Archivo, tendremos las opciones: Guardar fichero de Trabajadores, **Exportar a XML** y **Salir**. La opción de guardar volcará el contenido del ArrayList con la lista de los trabajadores a disco. Tanto al principio de la ejecución del programa, como cada vez que se guarde el fichero, la opción de exportar aparecerá inhabilitada, puesto que no habrá nada que guardar, y se habilitará cuando se haya producido algún cambio.



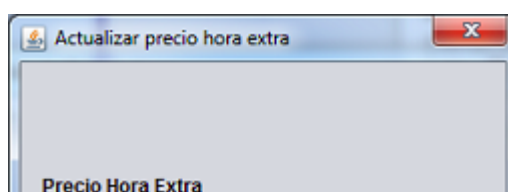
- ✓ En el menú Trabajadores, tendremos las opciones que ya conocemos, y que teníamos desde el primer programa:



- ✓ En el menú Utilidades, tendremos las opciones de Enviar correo electrónico a los trabajadores (lo comentamos más abajo) y la de Generar Fichero de Nóminas de los trabajadores:



- ✓ Cuando haya que introducir datos, los que hay que establecer como el sueldo fijo, cuota obrera, Trabajador, emplearemos un cuadro modal como el que se puede ver en la siguiente imagen:



Dialog box titled "Precio Hora Extra". It contains a text input field labeled "Nuevo precio hora extra:". Below the input field are two buttons: "Aceptar" and "Cancelar".

Actividad 2.- Añadir atributo nuevo a la clase Trabajador.

- ✓ Añadir un atributo de tipo **String** y nombre **email** a la clase **Trabajador**, para que lo hereden todos los constructores, métodos, opciones de los menús como las de importar, exportar, etc. sea necesario en definitiva para que la aplicación haga lo mismo que hacía hasta ahora, pero teniendo en cuenta la existencia de ese nuevo atributo. A modo sólo de ejemplo, la ventana de alta de trabajadores de campo de texto para recoger el valor del atributo **email** del nuevo trabajador que se está creando.
 - Se debe comprobar que el email introducido es válido. Para ello, crearemos un método `esEmailCorrecto(String email)` que comprobará que la dirección de correo está "bien formada" usando una expresión regular.
 - La expresión a usar para tener en cuenta todas las posibilidades de emails válidos es bastante complicada, pero lo que nos vamos a limitar a usar una que incluye la mayoría de los emails válidos, pero que puede intentar mejorar si ve que hay algún email correcto que no valida, o si ve que algún email no es válido. Dicha expresión regular es:

```
"^[A-Za-z0-9-\\+]+(\\.[A-Za-z0-9-\\+])*[A-Za-z0-9-\\+](\\.[A-Za-z0-9-\\+])*(\\.[A-Z
```

- El método anterior se usará para comprobar que un email es válido o no, de forma que el método `setEmail(String email)` solo asignará el email si éste es correcto según el método `esEmailCorrecto()`.
- Al dar de alta un nuevo trabajador fijo, se mostrará una pantalla similar a la siguiente (actualmente la pantalla es para trabajadores fijos, tendrá que mostrarse otra similar para trabajadores eventuales que permita recoger los valores para los atributos específicos de éstos últimos):

Form titled "Crear nuevo trabajador fijo". It contains the following fields and labels:

- NIF/NIE: [Text input field]
- Nombre: [Text input field]
- Fecha de nacimiento: [Text input field]
- Altura: [Text input field]
- Ocupación: [Text input field]
- Fecha contratación: [Text input field with value 2014]
- e-mail: [Text input field]
- N° de hijos: [Text input field]
- N° cuenta: [Text input field]
- Sueldo Fijo: [Text input field]

Buttons: "Guardar" and "Cancelar".

- Cuando se vaya a modificar un trabajador, se mostrarán los datos en una pantalla similar

en alta como en modificación, cuando se introduzca un dato incorrecto, como por ejemplo al darle al botón **Guardar**, se mostrará un mensaje de error similar al siguiente, que sería e-mail:



Actividad 3.- Mostrar Listados de trabajadores ordenados por distintos criterios.

- ✓ Para mostrar los trabajadores de la empresa, cuando se pulse en la opción de Listar Trabajador pantalla similar a la siguiente, donde como puedes ver, se podrá ordenar la lista de los trabajadores que se seleccione con los botones de radio.

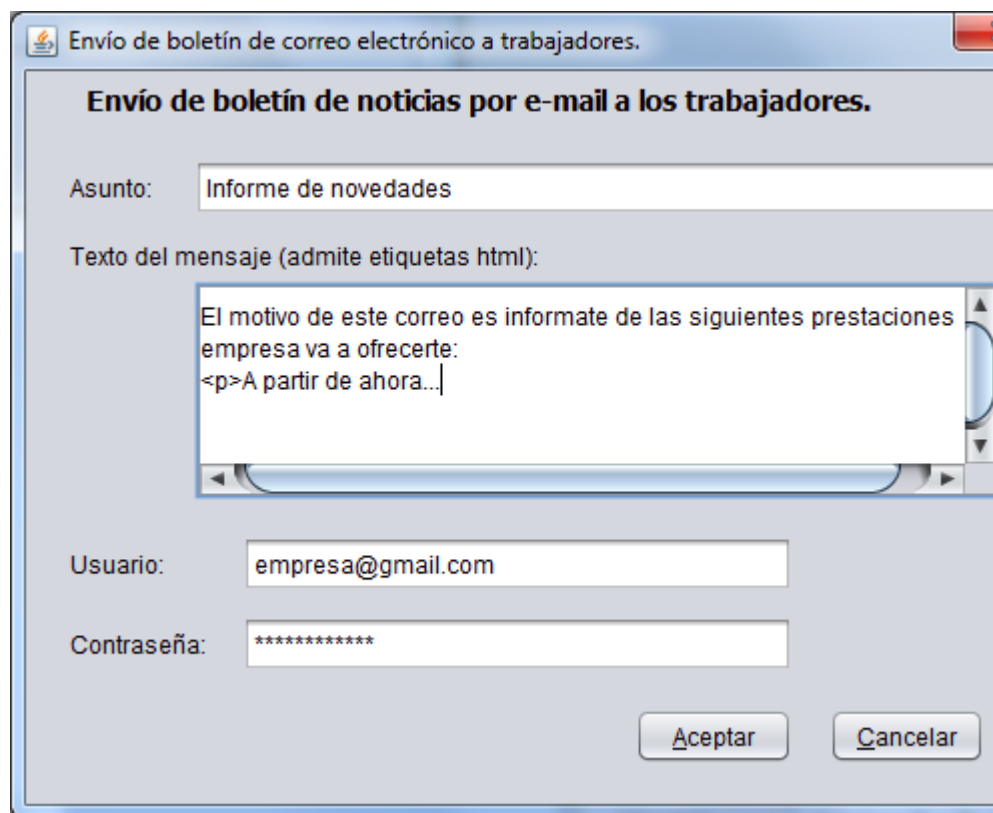
Totales por tipo de trabajador	
Total de Trabajadores en la empresa:	1
Nº de Trabajadores fijos:	1
Nº de Trabajadores eventuales:	0

Al estar sobre un trabajador y pulsar el botón de Ver Información, aparecerá la información del trabajador mediante un JOptionPane.

Actividad 4.- Añadir funcionalidad nueva a la aplicación: envío de boletín de noticias por email

trabajadores de la empresa.

- ✓ La opción del menú de Utilidades, sobre enviar correo, debe abrir una ventana en la que se pida necesarios para enviar el email, que en principio son:
 - **Asunto.** Se trata de indicar sobre qué trata el mensaje.
 - **Texto del mensaje.** Es el cuerpo del mensaje, el texto del boletín con las noticias de las informar. Se pueden introducir etiquetas `html`, (manualmente, eso sí, no es plan de comp momento) para darle formateo a la salida.
 - **Usuario.** Es la dirección de correo desde la que se envía el mensaje. Por defecto, tal y c imagen, debe proponer la dirección de correo de la empresa. No obstante, como se dese miembros de la Dirección de la empresa puedan hacer uso de esta aplicación para envia buzón personal a los trabajadores, se debe permitir introducir otra cuenta distinta. Para e las cuentas personales de los miembros de la Dirección tienen todas el mismo servidor d hemos configurado en el código.
 - **Contraseña.** Es la contraseña de usuario, para poder acceder al servidor de correo salie Debe introducirse en un `JPasswordField`, diseñado específicamente para introducir contri vea su contenido mientras se escriben, sustituyendo cada carácter por un asterisco, o alq se aprecia en la imagen.



- ✓ Se piensa que en el futuro puede ser deseable enviar emails a otro tipo de colectivos con lo que relaciona, como **Proveedores, Clientes**, etc. Es por eso que se ha pensado que sería deseable representar a aquellos colectivos que puedan recibir mensajes implementen el interface `Receptor` garantizar que implementará un método `enviarEmail(String usuarioRemitente, String password, textoMensaje)` de forma que podamos estar seguros de que si les enviamos ese método a un ot clase que implemente dicho interface, sabrá qué tiene que hacer para enviar el emai desde la di que se indique y para el que se ha suministrado la contraseña, con el asunto indicado y con el te del destinatario para el que se ha invocado al método. Queremos por tanto, que declares el inter `ReceptorDeMensajes` y que la clase `Trabajador` implemente dicho interface.

SOLUCIÓN DE LA TAREA.

A continuación, te puedes descargar una solución orientativa de la tarea.

[Solución de la tarea.](#) (944 KB)

Condiciones y términos de uso de los materiales

Materiales desarrollados inicialmente por el Ministerio de Educación, Cultura y Deporte y actualizados por el profesorado de la Junta de Andalucía bajo licencia Creative Commons BY-NC-SA.



Antes de cualquier uso leer detenidamente el siguiente [Aviso legal](#)

Historial de actualizaciones

Versión: 01.02.04	Fecha subida: 28/06/17	Autoría: José Javier Bermúdez Hernández
Ubicación: 5.10 (Debes conocer) Mejora (tipo 1): Uno de los vídeos (como añadir un JComboBox) no se encuentra disponible.		
Versión: 01.02.03	Fecha subida: 03/02/17	Autoría: José Javier Bermúdez Hernández
Ubicación: 5.8 Mejora (tipo 1): Cambiar imagen decorativa.		
Versión: 01.02.02	Fecha subida: 26/01/17	Autoría: José Javier Bermúdez Hernández
Ubicación: Apartado 1 Mejora (tipo 1): Actualizar la foro, pues la que hay ya no aparece en flickr		
Versión: 01.02.01	Fecha subida: 02/12/16	Autoría: José Javier Bermúdez Hernández
Ubicación: 4.6 Mejora (tipo 1): Añadir un Para Saber Más con los enlaces: http://www.arquitecturajava.com/java-8-lambda-expressions/ y http://www.dosideas.com/noticias/java/983-java-8-mas-alla-de-los-lambdas.html sobre las expresiones lambda en Java		
Versión: 01.02.00	Fecha subida: 09/11/15	Autoría: José Javier Bermúdez Hernández
Ubicación: 4.6 Mejora (tipo 1): Añadido un Para Saber Más con los enlaces: http://www.arquitecturajava.com/java-8-lambda-expressions/ y http://www.dosideas.com/noticias/java/983-java-8-mas-alla-de-los-lambdas.html sobre las expresiones lambda en Java Ubicación: Anexo		

Mejora (tipo 1): Eliminar el Anexo de licencias, poniendo las mismas al pie de cada imagen.

Ubicación: 5.1.- Contenedores

Mejora (tipo 2): Incluir un ejemplo o ejercicio resuelto que muestre cómo generar una ventana de tipo JDialog a partir de una ventana padre de tipo JFrame, pasándole además alguna información (por ejemplo a través del constructor mediante un ArrayList o HashMap). Aprovechar también para observar las diferencias de comportamiento entre que el JDialog sea modal o no.

Ubicación: Apartado 5.1

Mejora (tipo 1): Unificar el Para Saber más y Debes conocer, indicando además lo que es ventana modal y haciendo referencia al Anexo II para un ejemplo detallado de uso.

Ubicación: 4.5

Mejora (tipo 1): En los métodos de la interface MouseListener hay un mouseClicked y un par de KeyEvent donde debería de figurar mouseClicked y MouseEvent.

Ubicación: Apartado 5.4.- Contenedor ligero: JPanel.

Mejora (tipo 1): Cambiar el enlace del Para saber más a: http://programacion.net/articulo/swing_y_jfc_java_foundation_classes_94/20 o bien <http://www.cs.unicam.it/piergallini/home/materiale/gc/java/ui/swing/panel.html> u otro

Versión: 01.01.01

Fecha subida: 23/03/15

Autoría: José Javier Bermúdez Hernández

Ubicación: Apartado 4.4.

Mejora (tipo 1): El ejemplo enlazado como Código Java comentado de un oyente de teclado (22 KB) da un problema con la ç (C con cedilla). Al pulsarla la muestra bien, pero al soltarla no... Se soluciona cambiando para KeyReleased el método getKeyCode() por el método getExtendedKeyCode()

Versión: 01.01.00

Fecha subida: 24/03/14

Autoría: José Javier Bermúdez Hernández

Se ha modificado el título, se ha cambiado la unidad de la 7 a ser la 10, se han renombrado los recursos, se ha modificado la forma de enlazar los enlaces al glosario para que las definiciones vayan incorporadas en el .elp, se han añadido apartados para componentes Swing básicos que antes no se mencionaban (JInternalFrame, JToolBar, JProgressBar, JComboBox), y se han explicado mejor algunos de los que antes aparecían (JList...). Se ha rehecho el mapa conceptual haciendo que su descripción sea

Versión: 01.00.00

Fecha subida: 24/03/14

Autoría: MECD MECD

Versión inicial de los materiales, elaborados por el Ministerio de Educación. Antes correspondía a la unidad PROG07.