Tabla de contenidos

Introducción a la creación de interfaces gráficas de usuario (GUI) mediante Swing

1. Introducción

Introducción

*Una empresa moderna como **SI Andalucía**, debe adaptarse continuamente a las nuevas tendencias, y el cliente de software cada vez pide más facilidad de uso en las aplicaciones. Eso nos lleva inevitablemente a la creación de programas con entornos gráficos, amigables, intuitivos y generalmente más fáciles de usar, aunque como contraprestación tengamos la necesidad de ordenadores más potentes, pero esto cada vez es menor problema para las empresas.Es lo que **Carmen** le explica a **Víctor** una tarde al salir del trabajo, cuando él le pregunta por el resultado final de sus aplicaciones. **Víctor** entiende perfectamente que lo que le queda para ser un buen programador en Java, es precisamente esta parte, la **Programación Visual de Aplicaciones.***

Hasta ahora venimos haciendo programas, cada vez un poco más complejos, que poco a poco van requiriendo más y más **interactividad** con el usuario de la aplicación. Éste debe seleccionar la tarea a realizar entre todas las disponibles, o introducir los datos que la aplicación va solicitando. Todo ello lo hemos venido haciendo trabajando en "modo consola", mediante **menús de opciones**, que se escriben una y otra vez en la pantalla, y mediante la escritura de mensajes que nos solicitan la introducción de la opción elegida o de los datos que requiere la aplicación.

Pero ¿trabajan así todavía las aplicaciones reales que estamos acostumbrados a usar y a ver? ¿Crees que una empresa que te encargara un programa para una determinada tarea admitiría una aplicación que trabajara en modo consola?

La realidad es que no, o casi seguro que no. Puede que hasta ahora incluso te hayas sentido extrañado, porque esperabas hacer aplicaciones que trabajaran mediante un **interfaz gráfico de usuario** (Graphic User Interface o GUI). Dicho de otra forma, es posible que pensaras que en este módulo te íbamos a enseñar a realizar aplicaciones que funcionaran con sus típicas ventanas a las que tan acostumbrados estamos, con sus barras de herramientas y barras de menús y de título, con posibilidad de maximizarlas, minimizarlas, cerrarlas, moverlas por la pantalla, con cuadros de diálogo, botones de acción, casillas de verificación, y un largo etcétera de elementos que nos permitieran interactuar con la aplicación.

Pues no estabas equivocado. Ahora ha llegado el momento de empezar a **desarrollar aplicaciones con interfaz gráfica** de usuario en Java. Y aunque éste es un tema que se verá con más detalle en otro módulo profesional de este mismo ciclo formativo, concretamente el de **Diseño y Realización de Servicios de Presentación en Entornos Gráficos**, puedes estar seguro de que aquí aprenderás lo suficiente como para hacer aplicaciones completas y de aspecto bastante profesional.

Que el aspecto de esas aplicaciones sea más agradable visualmente o menos dependerá en gran medida de tus habilidades personales para el diseño, y de tu buen gusto.

Una guía para un buen diseño que debemos tener en consideración dice:

"Haz las cosas simples de forma sencilla, y las cosas difíciles hazlas posibles".

Otro enunciado interesante que debemos tener presente al hacer el diseño del interfaz gráfico de una aplicación es **el principio de asombrarse al mínimo**, que dice:

"No sorprendas al usuario"

Un buen diseño no es el aspecto que más importancia tiene por ahora en este módulo profesional, donde lo importante en estos momentos es conocer y aprender a usar los principales elementos gráficos que pueden incorporarse al interfaz gráfico de una aplicación.

Lo que más debemos valorar por ahora es la consecución de una aplicación completa, funcionalmente correcta, que haga todo lo que debe hacer en un tiempo razonable, de forma clara y sin cometer errores.

Conseguir que además sea una aplicación "amigable", o "visualmente atractiva" es muy importante, pero no es lo que pretendemos destacar por el momento.

2. AWT, Swing y las JFC

AWT, Swing y las JFC

***Carmen** recuerda el momento en que ella comenzó a programar aplicaciones para entornos gráficos, y lo que le costaba entender determinadas cosas, hasta que uno de sus compañeros le explicó que para llevar a cabo la **Programación Visual de Aplicaciones**, es preciso conocer bien el sistema operativo, el lenguaje y todas las librerías que facilitan el trabajo mediante rutinas ya programadas que resuelven los problemas más habituales de estas aplicaciones. Desde el momento que empezó a descubrir estos elementos, AWT, JFC o Swing que le daban dolor de cabeza, toda la programación en Java fue más sencilla, porque como le dijo su compañero; "aunque se trata de un extenso conjunto de programas, luego realmente siempre usas los mismos".*
*Ahora **Carmen** es una magnífica programadora en Java, ella lo sabe y se siente muy cómoda con estas herramientas, aunque eso no le debe impedir estar actualizada y conocer todas las novedades que aparecen relacionadas con la programación en Java.*

Programar aplicaciones con interfaz gráfico requiere por lo general hacer uso de **librerías** que nos aportan la funcionalidad básica para construir ventanas y añadirles componentes, etc.

Estas **librerías** frecuentemente son **extensiones** del lenguaje, que no forman parte del estándar del mismo, que han sido desarrolladas por separado y que se proporcionan también por separado. Y lo normal es que además deban ser pagadas por separado, como un complemento adicional al coste de adquisición del compilador básico.

De esta forma cada programador comprará aquellos paquetes de librerías que necesite, y no el total.

¿Es ese el caso de Java? ¿Qué herramientas nos proporciona el lenguaje Java para construir interfaces gráficos de usuario?

Uno de los motivos por los que se ha elegido Java como lenguaje para este módulo de programación, y por el que es elegido por muchas universidades y empresas, es que **Java proporciona de forma gratuita dentro del JDK que tú ya tienes instalado toda una amplia gama de librerías que puedes usar**. No tienes más que ver la documentación de la API para ver la enorme cantidad de paquetes que nos proporciona Java, cada uno de ellos útil para un tipo de desarrollo. Concretamente, también nos proporciona las librerías de clases básicas que nos permiten desarrollar aplicaciones con interfaz gráfica.

Esas librerías se engloban bajo los nombres de **AWT** y **Swing**, que a su vez forman parte de las **Java Foundation Classes o JFC.**

Veamos en los apartados siguientes a qué nos referimos con cada uno de esos nombres.

2.1. JFC o Java Foundation Classes

JFC o Java Foundation Classes

JFC son las siglas de Java Foundation Classes, que constituye una colección bastante completa de librerías de clases que proporciona el lenguaje Java, de forma gratuita, para permitir al programador disponer de una enorme funcionalidad lista para usar, sin tener que estar constantemente "reinventando" la rueda para resolver problemas ya resueltos.

Entre las clases de las JFC se incluye un importante grupo de elementos que ayudan a la construcción de interfaces gráficas de usuario (GUI) en Java.

Las JFC son las clases básicas, los cimientos sobre los que se construyen el resto de clases que se usan para crear componentes gráficos en las ventanas. Dicho de otra forma, las JFC son el conjunto de clases y librerías que Java ofrece de forma gratuita, que se instalan y que están disponibles junto al JDK.

Los elementos que componen las JFC son:

- **Componentes Swing**: que comprende 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 presume de ser multiplataforma, y se puede elegir el que se desee en cada momento.
- **Interfaz de programación Java 2D**: Permite a los programadores 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. Este término se utiliza aquí en contraposición a las aplicaciones Java que pueden ejecutarse en cualquier plataforma que incorpore la Máquina Virtual Java (JVM).
- **Soporte de impresión.**
- **Soporte de 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.

Es posible encontrar ejemplos de casi todos ellos (menos de los tres últimos) en los ejemplos que nos proporciona la web dedicada a ejemplos de **Java Web Start**, que no es más que una aplicación incluida en el JRE (entorno de ejecución Java), y que se instala automáticamente al instalar el JDK.

Java Web Start permite ejecutar aplicaciones de tipo "stand alone" de forma remota, facilitando la descarga e instalación, y haciéndolas transparentes al usuario, por lo que es una estupenda herramienta de distribución de aplicaciones Java.

Bastará con hacer clic en el enlace de descarga de la aplicación, para que ésta se descargue, se instale y se ejecute en el ordenador del cliente, sin necesidad de hacer nada más.

Es decir, en una página web podemos poner un enlace a una aplicación Java de tipo "stand alone" que queremos distribuir (que no es una applet, y que no está integrada en un marco de una página web, que no es una aplicación web) y basta con que el usuario seleccione ese enlace para que Java Web Start descargue y ejecute la última versión de esa aplicación en el ordenador del cliente.

De esta forma, el usuario siempre ejecutará la versión más reciente de la aplicación, ya que si se ha producido algún cambio, se volverá a descargar de nuevo. Si la aplicación necesita una nueva versión del entorno de ejecución Java, también será descargada e instalada para ejecutar la aplicación, pero sin alterar la instalación previa que tuviera el usuario en su equipo.

PARA SABER MÁS:
Si quieres ver una información completa sobre la tecnología Java Web Start, puedes consultar la información (en inglés) que proporciona la página de Sun. En ella encontrarás distintos enlaces, y entre ellos uno llamado Java Web Start White Paper, que te proporciona una guía sobre el uso y funcionamiento de Java Web Start. El enlace a la página de Sun sobre Java Web Start es el siguiente:
Java Web Start Technology
En el siguiente enlace encontrarás los ejemplos que te comentamos anteriormente, y que te proporcionan una muestra bastante buena de las posibilidades de Java para crear aplicaciones con interfaz gráfica de usuario. Basta con pinchar en el icono de cada una de las aplicaciones, o en su botón "Launch", para que se descarguen en tu ordenador y puedas usarlas. Observa que la primera vez hay que realizar la descarga, se tarda un poco más, pero que las siguientes veces, mientras que esas aplicaciones no se hayan modificado, la ejecución es inmediata. Merecen especial atención las aplicaciones "A Simple NotePad", que ejemplifica el uso de portapapeles y la posibilidad de impresión, "Draw", que ejemplifica el uso de gráficos en dos dimensiones, y sobre todo "The SwingSet Demo", que ejemplifica el uso y las posibilidades de la mayoría de los componentes Swing, además de la elección del aspecto (Look And Fell).
El enlace a todos estos ejemplos es el siguiente:
Demos de Java Web Start que ejemplifican las posibilidades GUI de Java

2.2. AWT o Abstract Window Toolkit (I)

AWT o Abstract Window Toolkit (I)

Evidentemente, como se deduce del título de este apartado, AWT son las siglas que usaron los creadores de Java en sus primeras versiones para nombrar el **conjunto de paquetes y clases destinadas a proporcionar un Kit de herramientas para crear ventanas abstractas** que pudieran incluirse en las aplicaciones que se desarrollaran con el lenguaje.

¿Qué queremos decir cuando hablamos de ventanas abstractas?

Nos referimos a ventanas genéricas, que proporcionan las herramientas básicas necesarias para que el programador usuario las personalice, dándoles el aspecto y la funcionalidad que desee para una aplicación concreta.

¿Qué características tienen los componentes AWT para desarrollar aplicaciones gráficas?

- **Los desarrolladores del AWT original usaban una ventana del sistema operativo para cada uno de los posibles componentes**. De esta forma, cada cuadro de texto, cada casilla de verificación, cada botón, etc. tenía su propia ventana, que debía ser gestionada por el sistema operativo. Esto significaba que cualquier aplicación tenía que gestionar un enorme número de ventanas, y por ello cualquier aplicación gráfica consumía una gran cantidad de recursos del sistema (memoria y tiempo de CPU).

La creciente complejidad de las aplicaciones con interfaz gráfico hacía insostenible ese modelo, ya que el coste en eficiencia de las aplicaciones era excesivo.

- **Por otro lado, las clases AWT estaban desarrolladas usando código nativo** (es decir, código asociado a plataformas concretas). **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 dónde se pretendía usar AWT**. Como consecuencia, **AWT es una librería con una funcionalidad muy pobre**.
- **El modelo de programación de AWT inicial no era ni siquiera orientado a objetos (aunque sí lo es el actual, desde Java 1.1).**
- **AWT sigue siendo imprescindible, ya que todos los componentes Swing se construyen haciendo uso de clases de AWT**. De hecho, como puedes comprobar en la 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`.

La imagen siguiente muestra gráficamente cómo aunque nuestra aplicación esté hecha con Swing, éste se construye sobre AWT, que a su vez se construye sobre JFC.

- **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`.**

2.3. AWT o Abstract Window Toolkit (II)

AWT o Abstract Window Toolkit (II)

De todas estas características de AWT, podemos concluir que fue la primera forma de construir las ventanas en Java, pero que limitaba la portabilidad, restringía la funcionalidad y requería demasiados recursos.

Por tanto, era necesario mejorar esas características, y 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 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. Por tanto, Swing es el fruto de la colaboración entre Netscape y Sun.**

Teniendo en cuenta que la primera versión de AWT se desarrolló en un mes, a pesar de que es un ejemplo de asombrosa productividad del equipo de desarrolladores, se explican algunas de sus carencias.

PARA SABER MÁS:
En este enlace podrás profundizar sobre las aplicaciones gráficas en Java
Aplicaciones Gráficas
En este enlace, en forma de glosario, encontrarás una explicación muy interesante sobre qué son los controles ActiveX
Controles ActiveX

Cada uno de los posibles componentes AWT tiene una clase del paquete `java.awt` asociada.

A continuación te proporcionamos una lista de las clases AWT más populares, aunque la lista no es ni mucho menos completa:

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.**

2.4. Swing

Swing

En el apartado anterior hemos mencionado los grandes inconvenientes de usar directamente componentes AWT en nuestras aplicaciones

- Dificultan la portabilidad, al estar implementados con código nativo
- Restringen la funcionalidad
- Consumen excesivos recursos.

También mencionábamos que Swing nace con la sana intención de superar esos problemas, y de hecho lo consigue.

¿Pero qué es Swing? ¿El nombre significa algo especial?

En cuanto al nombre no significa nada en especial. A diferencia de AWT, Swing no son las siglas de nada. Al igual que al lenguaje le llamaron Java porque los comienzos se produjeron en reuniones que tuvieron lugar en un café, Swing fue el nombre que le dieron al proyecto de "mejora y actualización" de las posibilidades gráficas de Java. El porqué de ese nombre no nos consta. Seguramente hace referencia a los gustos de sus miembros por ese tipo de música...

En cuanto a lo que es Swing. A grandes líneas, podemos decir de Swing que:

- **Es una librería de clases dirigidas específicamente al diseño de interfaces gráficas de usuario**.
- Para ello **las clases Swing implementan la funcionalidad básica, de una forma flexible (configurable por el programador)**
- **Es independiente de la arquitectura** (tecnología 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.**
- **Por cada componente AWT (excepto Canvas) existe un componente Swing equivalente**, **cuyo nombre empieza por J, pero reescrito enteramente en Java, y 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.
- **Swing aumenta el número de componentes que se pueden usar respecto a AWT**
- **Los componentes Swing no necesitan una ventana propia del sistema operativo cada uno**, si no 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
- Por **ello las clase 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.**

En este módulo profesional nos ocuparemos de aprender a desarrollar aplicaciones Java usando directamente Swing, sin ver previamente AWT, sobre el que basta con que sepamos de su existencia y que Swing está construido sobre AWT.

A continuación puedes encontrar un enlace a una **tabla** con la lista de las clases Swing más habituales.

- Observa que aparecen los mismos componentes que en AWT pero con los nombres comenzando por J, y muchos más componentes que no estaban en AWT, ya que Swing permite mucha más funcionalidad.
- La lista siguiente no es tampoco exhaustiva, pero incluye casi la totalidad de elementos que necesitaremos por ahora.
- La mayoría de ellos puedes verlos en la aplicación "The SwingSet Demo" que mencionábamos en el Para Saber Más, del apartado 2.1. de esta unidad. De hecho, las imágenes están sacadas de esa aplicación, que es el ejemplo desarrollado por Sun para mostrar de forma más o menos completa las posibilidades de Swing.
- También debes tener en cuenta que las imágenes están obtenidas eligiendo el aspecto (LookAndFeel) multiplataforma propio de Java. Puedes ver el aspecto "nativo" que tendrían esos componentes en plataformas Windows o Unix (aspecto Motif) sin más que seleccionar la opción adecuada en el menú Look&Feel de la propia aplicación.

Tabla con la lista de las clases Swing

En esa lista no aparece un componente `JCanvas`, que no existe. La razón 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.

PARA SABER MÁS:
Podrás ampliar tus conocimientos de Swing visitando los siguientes enlaces, así como encontrar ejemplos de sus aplicaciones.
Este enlace tiene una guía de Swing presentado en forma de índice. Es muy interesante como primera aproximación a Swing.
Indice de Swing
En estos dos enlaces encontrarás una muy buena profundización sobre Swing, los dos son muy interesantes. El interés de la versión en inglés radica en que se trata del tutorial original elaborado por la propia empresa desarrolladora de Java.
Tutorial en español de Swing
Tutorial de Sun en inglés sobre Swing

Autoevaluación

3. Paquetes Swing

Paquetes Swing

__Carmen__ tiene muchas tareas asignadas en __SI Andalucía__. Una de ellas es la de ayudar a __Víctor__ a formarse como programador en Java, con el fin de que colabore en la elaboración de aplicaciones de la empresa. La verdad es que a __Carmen__ esta tarea le agrada bastante, especialmente porque fue el propio __Víctor__ quien sugirió que fuera ella la que le guiase en su formación como programador, diciendo que... "ella sabe cómo explicarme las cosas, para que yo las entienda".
Mientras se dirige al trabajo, va pensando cómo puede explicarle a __Víctor__ la incorporación de los paquetes Swing para que un programa pueda utilizar todos los componentes gráficos. Finalmente ha decidido tratarlo como lo que realmente son, conjuntos de objetos, cada uno con sus atributos (propiedades) y métodos (eventos). Está convencida que __Víctor__ va a entenderlo muy bien en cuanto vea un ejemplo.

Recuerda que los paquetes son una unidad de organización del código en Java. **Un paquete no es más que una carpeta en la que se guardan una serie de clases que tienen alguna relación entre sí**. Por eso todas las clases relativas a Swing se encuentran dentro de una misma carpeta, es decir un mismo paquete, que a su vez las contiene clasificadas en subpaquetes según su utilidad.

Todas las clases Swing están recogidas dentro del paquete **javax.swing** y sus subpaquetes. La lista completa es la siguiente:

- **javax.swing**
- **javax.swing.border**
- **javax.swing.colorchooser**
- **javax.swing.event**
- **javax.swing.filechooser**
- **javax.swing.plaf**
- **javax.swing.plaf.basic**
- **javax.swing.plaf.metal**
- **javax.swing.plaf.multi**
- **javax.swing.table**
- **javax.swing.text**
- **javax.swing.text.html**
- **javax.swing.text.html.parser**
- **javax.swing.text.rtf**
- **javax.swing.tree**
- **javax.swing.undo**

Dependiendo de los componentes que incluya nuestra aplicación, y de lo que queramos hacer con ellos, deberemos asegurarnos de que nuestra aplicación contenga las sentencias `import` necesarias para todos los paquetes de los que usemos alguna clase.

Así, por ejemplo, en cualquier aplicación Swing que construyamos tendremos que incluir al menos la sentencia:

```
import javax.swing.*;
```

Afortunadamente para nosotros, **al trabajar con un entorno integrado de desarrollo, dispondremos de un diseñador,** en el que iremos creando las ventanas, y añadiéndoles componentes de una forma gráfica, y **será el propio entorno el que genere gran parte del código Java necesario** para implementar esos componentes. Entre otras cosas, **también se encargará de añadir las sentencias import necesarias** para todos esos componentes, o en su defecto, cada vez que se use una clase, añadirá la ruta del paquete en el que se encuentra. Este parece ser el camino preferido por NetBeans. Esto lo veremos más claramente cuando creemos una aplicación de ejemplo.

4. Contenedores Swing

Contenedores Swing

*La primera lección que va a recibir **Víctor** sobre programación visual con entornos gráficos se la va a dar **Carmen**, que le explica que al principio parece que todo es muy fácil sobre el papel, pero a la hora de crear un proyecto con componentes gráficos la cosa cambia. No sabes situarte, no comprendes lo que está pasando porque todo cambia y tienes que decidir hasta las ventanas. Pero **Carmen** lo tiene claro, hay que empezar por los contenedores que como su propio nombre indica se emplean para ubicar el resto de componentes.*

En los apartados anteriores nos hemos limitado a:

- Hacer un poco de historia justificando la necesidad de usar componentes gráficos en nuestras aplicaciones.
- Indicar que Swing es la librería de Java que nos ofrece gran cantidad de componentes, con una amplia funcionalidad y versatilidad, y mejorando la eficiencia.
- Mostrar 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 de una idea de cual es su aspecto.

Hemos visto que esos componentes no son nada especialmente raro. Son el tipo de controles que estamos acostumbrados a usar en la mayoría de las aplicaciones. Pero nuestro conocimiento de ellos debe ir un poco más allá de su uso. Ahora debemos aprender a incluirlos dentro de una aplicación, y a asociarles la funcionalidad que deseamos a cada uno de ellos.

Y eso nos lleva a la necesidad de que cada aplicación "contenga" de alguna forma esos componentes. ¿Cómo podemos hacerlo? ¿Qué componentes se usan para contener a los demás?

Esa es la función de un grupo de componentes especiales de Swing, que se llaman contenedores Swing.

Swing proporciona dos tipos de elementos contenedores:

- **Contenedores de alto nivel. (Marcos: `JFrame y Jdialog` para aplicaciones)** También **`JApplet`**, para applet, pero por ahora nos centramos en las aplicaciones **stand alone**.
- **Contenedores de bajo nivel. (Paneles: `JRootPane , Jpanel`)**

4.1. Contenedores Swing de alto nivel

Contenedores Swing de alto nivel

Cualquier aplicación típica a las que más o menos estamos acostumbrados siempre 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 la podemos considerar como un marco dentro del cual podemos ir colocando el resto de componentes que necesitemos (menú, barras de herramientas, barra de estado, botones, casillas de verificación, cuadros de texto, etc.)

Puestos a poner un ejemplo de ventana principal con el que puedes estar más o menos familiarizado, te ponemos la ventana principal del bloc de notas. Es una ventana muy simple, en la que vemos los elementos comentados: Marco delimitado por los bordes de la ventana, botones de minimizar-maximizar-cerrar en la barra de título, y aparentemente poco más. Ese es el "Contenedor de alto nivel de la aplicación"

Pero incluso en esta ventana simple de una aplicación simple, ese "poco más" incluye otros muchos elementos: unas barras de desplazamiento, una barra de menú con distintos menús que nos abren otras ventanas y un área de texto en la que mostrar el trabajo realizado (en este caso, el texto que queremos escribir)

- **Esa ventana principal o marco sería el contenedor de alto nivel de nuestra aplicación.**
- **Toda aplicación Java, sea del tipo que sea, tiene al menos un contenedor de alto nivel.**
- **Los contenedores de alto nivel son componentes "peso pesado",** ya que extienden directamente a (son subclases de) una clase similar de AWT. Es decir, realmente **necesitan crear una ventana del sistema operativo independiente para cada uno de ellos**.
- **El resto de componentes, serán de "peso ligero"**, es decir, que **no tienen su propia ventana del sistema operativo, si no que se dibujan en su objeto contenedor.**

En Java existen cuatro tipos de contenedores de alto nivel, que te mostramos en la siguiente tabla:

Nombre de la clase Swing para el tipo de contenedor de alto nivel	Descripción
`JFrame`	**Crea un marco, una ventana principal para la aplicación**, que consta de Barra de título, con su menú de control, los botones de minimizar, maximizar/retaurar y cerrar, y con los bordes que la delimitan. También tiene alguna funcionalidad asociada, tal como cerrar la ventana, o cambiar su tamaño o posición mediante el ratón.
`JDialog`	**Crea una ventana secundaria, un cuadro de diálogo que se abre para interactuar con el usuario**, mostrándole información, o solicitándosela.
`JApplet`	**Crea un marco para una applet, es decir, para la ejecución de una aplicación Java integrada en una página web HTML, en una ventana del navegador**. JApplet proporciona el contenedor de alto nivel o ventana principal para este tipo de aplicaciones. No lo vamos a usar en nuestros ejemplos.
`JWindow`	Un JWindow es un contenedor de alto nivel que puede ser mostrado en cualquier parte del escritorio del usuario. No tiene ni la barra de título, ni ningún botón de manejo de ventana, ni ninguna funcionalidad asociada, pero es un elemento del escritorio del usuario y puede existir en cualquier parte del mismo. No lo vamos a usar en nuestros ejemplos.

En nuestro caso, por ahora usaremos como contenedor para nuestras aplicaciones "stand alone" principalmente **`JFrame`**, para la ventana principal, y **`JDialog`** para algunas de las ventanas secundarias que abrirá la aplicación.

Por ahora no vamos a hablar de aplicaciones web, por lo que dejamos aparcado el uso de **`JApplet`** para crear applets.

`JFrame`, al igual que todos los otros contenedores de alto nivel, **contiene un objeto `JRootPane` como su único componente hijo**. Ese objeto es un panel encerrado en los límites de la ventana, y podríamos representarlo como el panel de contenido (content pane), es decir, **en él se "colgarán" todos los componentes mostrados por la ventana `JFrame`, salvo los elementos del menú.** Por tanto ese panel de contenido existe por defecto, sin más que crear un nuevo objeto **`JFrame`**, y será un contenedor de bajo nivel sobre el que colocar el resto de componentes de la ventana.

Esto quiere decir que podemos añadir componentes a la ventana directamente usando el método **`add()`** y eliminarlos usando el método **`remove()`**, y que ese panel de contendido tendrá establecido por defecto algún gestor de distribución (Layout Manager) que indicará donde situar en la ventana cada nuevo componente.

Por ejemplo, supongamos que quiero añadir un botón llamado **`aceptar`** (declarado de tipo **`JButton`**) a una ventana llamada **`ventanaPrincipal`** (de tipo **`JFrame`**). La sentencia siguiente se encarga de esa tarea.

```
ventanaPrincipal.add(aceptar);
```

Afortunadamente, como verás en las presentaciones del apartado 6, todo esto es mucho más fácil, ya que **el diseñador incorporado en el IDE se encarga de generar el código necesario. Nosotros sólo tenemos que pinchar sobre el icono del componente que queremos añadir, escoger el gestor de distribución (Layout) que deseamos, y arrastar el componente hasta el panel de contenido de la ventana que estamos creando, que es el área de trabajo del diseñador. Éste se encarga de generar el código Java necesario.**

DEMO: Mira cómo se crea una aplicación con ventanas en NetBeans

4.2. Contenedores Swing de bajo nivel (I)

Contenedores Swing de bajo nivel (I)

En el apartado anterior hemos visto que los componentes de alto nivel no son más que un **marco**, que podemos mover, cambiar su tamaño o cerrar. Pero necesitamos que ese marco contenga un panel. Al igual que para un tablón de anuncios necesitamos que además del marco, haya un panel de corcho sobre el que colgar los anuncios, en las ventanas de una aplicación, además del marco de la ventana, necesitamos que esa ventana contenga un panel sobre el que poder "colgar"(dibujar, realmente) el resto de componentes. Ese panel de fondo de la ventana, que justamente se llama así, panel, se moverá con la ventana, "llevando consigo" el resto de componentes que se le han añadido.

Esos paneles donde podemos colgar otros componentes como botones, cuadros de texto, etiquetas, etc. son por tanto contenedores, ya que su función es contener a otros elementos. Pero a diferencia de los marcos y ventanas (`JFrame`, `JDialog` y `Japplet`), **los paneles no necesitan su propia ventana del sistema, por lo que los llamamos contenedores de bajo nivel.**

En la práctica, **al crear el marco de la aplicación con la clase `JFrame`, como ya vimos en el apartado anterior, también se crea un contenedor de bajo nivel de la clase `JRootPane`**.

Es sobre éste sobre el que se cuelgan los demás componentes. De hecho, `JRootPane` es la clase que gestiona el aspecto de los objetos del `JFrame`. Concretamente, el panel raíz (rootpane) de un objeto `JFrame` contiene varios objetos, que son como varias capas. A continuación vamos a explicar la teoría relativa a la estructura de esas "capas". En el apartado 6 veremos de forma práctica en un ejemplo su creación y uso. Un `JRootPane` está compuesto por:

- `glassPane` (objeto de la clase `Component`) **Es una especie de panel o capa transparente, que siempre se sitúa encima de todos los demás para interceptar los movimientos del ratón**. Es sobre este panel o capa sobre el que se desplaza el dibujo del puntero del ratón. **Ocupa todo el área visible de la ventana.**
- `layeredPane` (objeto de la clase `LayeredPane`) **Es el encargado de gestionar el panel de contenido y la barra de menú**, en caso de que la aplicación tenga una barra de menú. Se sitúa detrás del `glassPane`, y sobre él estará la capa de menús, de forma que cuando se despliegue un menú se dibuje por encima del resto de componentes de la ventana. También sobre él está el panel o capa de contenidos, que contendrá la mayoría de los componentes de la ventana con los que interactúa el usuario. **El `layeredPane` ocupa también todo el área visible de la pantalla**.
- `menuBar` **Es opcional, y puede no estar**. Aunque le hemos llamado `menuBar`, realmente es el usuario el que le asignará nombre a este objeto, y puede llamarse de cualquier otra forma. **Si la aplicación dispone de barra de menús, será un objeto de tipo `JMenuBar`. Podemos considerarlo como otra capa, que está por encima de la capa de contenidos de la ventana, y que por tanto, siempre se ve dibujada encima del resto de componentes de la ventana.** Por eso, al desplegar un menú, éste se ve por encima del resto de componentes de la ventana. De esta forma, el menú siempre está visible. Es una parte del `layeredPane`, y se sitúa en la parte superior del mismo, junto al borde superior.
- `contentPane` Es un objeto de la clase `Component`. **Es sin duda el panel con el que más interactuaremos. Generalmente los controles y los gráficos se sitúan en el `contentPane`. Es también una parte del `layeredPane`, que ocupa todo el área visible de la ventana, a exepción de lo que ocupe `menuBar`, si existe**. Desde el programa, podemos acceder al `contentPane` de una ventana que hayamos creado mediante el método `getContentPane()` disponible para las clases `JFrame` y `JApplet`. También es posible crear un panel, de cualquier tipo (basta con que sea una subclase de `Component`) lleno de componentes que queremos que tenga nuestra aplicación, y decirle mediante el método `setContentPane()` que queremos que ése sea el panel de contenido de nuestra ventana. En tal caso, ese panel normalmente lo crearíamos como un objeto de la clase `JPanel`.

La siguiente figura muestra la estructura del `JRootPane` de cualquier ventana.

Bien, esto parece bastante más complejo que el panel de corcho de un tablón de anuncios, ¿verdad?

Pero no te asustes. Hasta ahora te estamos contando la estructura que tienen los componentes Swing, que te ayudará a comprender porqué tienen el aspecto que tienen, y porqué las ventanas se comportan de una determinada manera.

4.3. Contenedores Swing de bajo nivel (II)

Contenedores Swing de bajo nivel (II)

Pero por suerte para nosotros, **Java se pensó para facilitar las cosas al programador, y no para complicárselas. El hecho de que ese panel de contenidos de la ventana sea complejo, sólo significa que ya se complicaron la vida los desarrolladores del lenguaje para que no nos la tengamos que complicar nosotros ahora al diseñar nuestras aplicaciones.**

En la mayoría de los casos, será suficiente con que sepas que:

- En cualquier ventana que crees va a existir un `contentPane`, que es el panel donde puedes colocar todos los componentes que deseas que tenga tu aplicación
- Si deseas introducir un menú, éste va a ir en su propio panel, que siempre va a estar visible. Es decir, por más componentes que añadamos a nuestro panel de contenidos, ninguno se va a dibujar encima del menú, que tiene su trozo independiente de ventana reservado.

Además, como veremos en el apartado 6, crear la ventana, y añadirle componentes es sumamente sencillo con la ayuda del diseñador que trae incorporado cualquier entorno integrado de de desarrollo más o menos actual.

Autoevaluación

5. Arquitectura MVC (Modelo-Vista-Controlador)

Arquitectura MVC (Modelo-Vista-Controlador)

*Al principio **Víctor** no estaba muy convencido de que le gustara la programación con Java, pero una vez que ha visto la programación visual, está entusiasmado y piensa que ya es capaz de crear una aplicación con una interfaz más o menos compleja que funcione adecuadamente ante las acciones de los usuarios. Pero cuando **Carmen** comienza a explicarle el MVC, **Víctor** no entiende nada. Él sólo necesita conocer las necesidades del cliente para diseñar las pantallas y programar las acciones.*
***Carmen** le explica que eso no es la programación Visual, a estas alturas ya debería saber que al construir una aplicación nos basamos en modelos reales, normalmente métodos de trabajo y modos de actuar ante determinadas situaciones. Eso es precisamente lo que se persigue al aplicar técnicas de programación, ajustarse al modelo real y conseguir una representación fiel del mismo, al menos en su comportamiento.*

Seguramente estarás deseando ver de forma práctica lo que se ha explicado en el apartado anterior, pero todavía quedan algunos aspectos teóricos sobre la forma que Swing tiene de hacer las cosas, que conviene revisar antes de empezar con las prácticas.

Comprender esta arquitectura Modelo-Vista-Controlador (MVC) de los componentes Swing, es fundamental para hacer un buen uso de los mismos.

La arquitectura MVC significa que:

- **El modelo de un componente está donde están almacenados sus datos.**

Por ejemplo, en una lista desplegable (`JComboBox`), los datos que deben mostrarse al desplegar la lista, pueden estar almacenados en un array de String, por ejemplo. Ese array de String con los elementos a mostrar en la lista, es el modelo de ese JComboBox. También puede ser el estado de un botón (pulsado (clicked), soltado (released), bajo el ratón (entered), etc.

- **La vista es la representación en pantalla del componente, es decir, es justamente el "dibujo" del componente que nosotros vemos en la ventana de la aplicación.** Lo que muestra la vista es justamente la información que contiene el modelo. En el caso de la lista, es el dibujo de la lista desplegable mostrando los elementos contenidos en el array de String que usamos como modelo. En el caso del botón, podemos tener un botón representado de forma diferente, (con distinta forma o color o texto o imagen) para cada uno de sus distintos estados, y sigue siendo el mismo botón.
- **El controlador es la parte del componente que gestiona los eventos, como los clicks del ratón**. Piensa que los componentes que incluimos en la ventana normalmente sirven para interactuar con el usuario (botones para realizar acciones, casillas de verificación para activar o no determinadas opciones, cuadros de texto en los que escribir y recoger información, etc.). **Cada una de las interacciones posibles con el usuario de un componente es lo que denominamos evento.** Y esos eventos deberán ser "capturados" cuando se produzcan y "procesados" para que la aplicación haga lo que debe hacer cuando se producen. Es el usuario el que tiene que escribir el código que indique lo que tiene que ocurrir cuando se pulsa un botón, o cuando una casilla de verificación está activa, o cuando se pulsa una opción del menú. Ese código que escribe el usuario para indicar lo que tiene que hacer la aplicación ante un evento producido para un componente concreto es el controlador del componente.

Esto tiene la ventaja de que al separar la vista del modelo, por ejemplo, podemos cambiar el aspecto de un componente según los datos que almacene en su modelo. Esto hace posible en Swing, por ejemplo, que tengamos distintas representaciones gráficas de un mismo botón, algo que no era posible en AWT.

Realmente, los aspectos relacionados con la programación guiada por eventos, y el manejo de los mismos, son materia de unidades siguientes. Por ahora basta con que entiendas cómo se estructuran las aplicaciones gráficas, y que tengas una visión más o menos clara a través de un ejemplo.

6. Ventanas, marcos, menús, paneles y bordes. (JFrame, JDialog, JPanel, Border, BorderFactory)

Ventanas, marcos, menús, paneles y bordes. (JFrame, JDialog, JPanel, Border, BorderFactory)

__Carmen__ decide que es el momento adecuado para que __Víctor__ empiece a ver ejemplos del funcionamiento de los componentes anteriores, porque parece que se está desanimando y aburriendo ante sus explicaciones. Para ello lo ideal es comenzar por los componentes __contenedores__, sobre los que se ubican los controles que verá el usuario y sobre los que va a actuar en un momento dado para conseguir aprovechar la aplicación.
__Víctor__ se sorprende al ver la cantidad de cosas que tiene que hacer el programador y que luego no se aprecian cuando la aplicación está funcionando. __Carmen__ le explica que todo esto es la preparación de la aplicación y que para que todo funcione correctamente no se puede improvisar, sino que debe estar perfectamente planificado y saber qué hacer (y cómo hacerlo), antes de sentarse ante el ordenador para programar.

En este apartado pretendemos darte un ejemplo que te ayude a entender mejor los conceptos de los apartados anteriores, y en parte de los siguientes. No se ha incluido un ejemplo en cada uno de los apartados anteriores porque no son aspectos separados, si no fuertemente ligados, y por eso es preferible presentarlos formando parte de un mismo ejemplo.

Los primeros ejemplos que vamos a abordar para construir una primera aplicación, van a ser muy simples:

- El primero consistirá en crear una aplicación que cuente el número de pulsaciones que se realicen y que ponga a cero el contador de dichas pulsaciones, mostrando la acción que se está realizando en un panel inferior, "panel de mensajes", separado del "panel de operaciones".
- También te proporcionamos una serie de presentaciones que te permitirán ver el proceso completo de creación de esta aplicación. Al ser un ejemplo tan simple, podremos centrarnos en comentar los aspectos de dotar a la aplicación de interfaz gráfico con Swing.
- Finalmente, también te proporcionamos el código de una aplicación sólo un poco más elaborada, para dotar de interfaz gráfica al programa que calculaba las soluciones de una ecuación de segundo grado. Así podrás comparar algunos aspectos en los dos ejemplos.

Vamos a ver entonces lo que incluirá cada uno de estos ejemplos:

- Usará como marco principal un `JFrame`.
- Veremos que puesto que automáticamente dispone de un panel de contenido, podemos añadirle componentes sin más desde el diseñador. Es el caso de cualquiera de los botones, que se añaden directamente al panel de contenido que tiene por defecto la ventana `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 usaremos 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 los ejemplos hemos elegido `TitledBorder` para todos los paneles que hemos añadido.
- En cada uno de esos paneles incluiremos las etiquetas y cuadros de texto que necesitamos para introducir y mostrar la información adecuadamente.
- Aunque es motivo de un análisis más detallado en el comienzo de la unidad 18, veremos como se puede elegir el tipo de gestor de distribución (layout) que deseemos para distribuir los componentes dentro de cada panel. Elegiremos null, que nos permite situar los componentes justamente en el lugar y con el tamaño que deseemos.
- Añadiremos una barra de menú, que será un elemento de tipo `JMenuBar`, con un primer menú llamado Archivo (de tipo `JMenu`) cuyas únicas opciones serán Salir de la aplicación, y abrir un cuadro de diálogo de tipo `JDialog` con información sobre la aplicación (Ambas serán elementos de tipo `JMenuItem`) Otro segundo menú tiene dos opciones que son excluyentes, (dos `JRadioButtonMenuItem` pertenecientes al mismo `JButtonGroup`) que servirán para cambiar el LookAndFeel de la aplicación. No obstante, el estudio y uso más detallado de menús se abordará al principio de la unidad 19.
- Crearemos un cuadro de diálogo con `JDialog` para mostrar información sobre las opciones de la aplicación y para proponer datos de prueba que permitan comprobar todos los casos posibles en la ejecución de la aplicación. Toda esa información está dentro de un área de texto (`JTextArea`) que se ha colocado sobre un `JScrollPane`, que es un panel que proporciona automáticamente barras de desplazamiento horizontales y verticales, si el tamaño de la ventana y el texto contenido en el área de texto lo requieren.
- Para conseguir que la aplicación haga algo, necesariamente vamos a tener que incluirle captura de eventos, pero eso también es materia de estudio de la unidad 18. Aunque con el diseñador se hace de forma bastante transparente al usuario.
- También ejemplificamos el uso de la clase `NumberFormat` para darle formato a números enteros. Esta clase realmente no es un elemento del interfaz gráfica, y no forma parte de las librerías Swing, pero con el uso de cuadros de texto, se hace evidente la necesidad de mostrar los números con un máximo de cifras decimales.
- También se han usado métodos para mejorar el aspecto y para usar los demás componentes que se han añadido, pero que serán objeto de estudio en las próximas unidades. Por ejemplo, a todos los componentes se les ha activado la propiedad ToolTipText, a todos los botones y opciones de menú, se les ha activado la propiedad Mnemonic, a los menús se les han habilitado aceleradores o atajos de teclado, mediante la propiedad Accelerator. También se les ha cambiado el texto, el color, el tamaño o el tipo de letra con las propiedades Text, Foreground y Font.

En las siguientes presentaciones podrás encontrar el proceso completo de creación de la aplicación PrimeraVentana.

DEMO: Visualiza cómo añadir una barra de menú a nuestra aplicación

DEMO: Ahora vemos cómo añadir una panel adicional

DEMO: Y ahora, crearemos un grupo de botones

DEMO: Aquí añadiremos una acción a la opción "Salir" del menú

DEMO: Y aquí, creamos una ventana nueva para la información "Acerca de..."

DEMO: Para terminar, hacemos que los botones de cambio de aspecto funcionen correctamente

A continuación tienes el enlace a la carpeta del proyecto que contiene esta pequeña aplicación, Primeraventana, usada en las presentaciones anteriores, para que puedas verla en funcionamiento.

Descarga el proyecto PrimeraVentana

También incluimos el código del ejemplo desarrollado en la aplicación EcuacionSegundoGrado.

Descarga el proyecto EcuacionSegundoGrado

7. Manejo del aspecto y comportamiento (LookAndFeel)

Manejo del aspecto y comportamiento (LookAndFeel)

*Cuando **Carmen** llega una mañana a la oficina, encuentra a **Víctor** trabajando ante el ordenador con los componentes de Swing que estuvieron viendo el día anterior. Le han quedado unas ventanas magníficas, que ha decorado con buen gusto, para las que dice que ha empleado más de tres horas. Dice orgulloso que estuvo trabajando anoche en casa hasta las cuatro de la mañana, y se muestra muy satisfecho de su trabajo.*
***Carmen** le felicita y le muestra su admiración por el buen trabajo realizado, pero le explica que en la empresa deben hacerse las cosas de modo que se aproveche el tiempo al máximo y normalmente no pueden permitirse el lujo de dedicar cuatro o cinco horas a "decorar" una ventana, para eso ya existen soluciones en Java que permiten adaptar el aspecto y el comportamiento de los componentes y conseguir un aspecto más corporativo de las aplicaciones. Y **Víctor** lejos de desanimarse, se interesa por estas soluciones de Java, pensando en las nuevas posibilidades que se le presentan.*

Ya hemos mencionado en los apartados anteriores que una de las innovaciones que introduce Swing frente a AWT es la posibilidad de seleccionar el aspecto de la aplicación. A esta característica en Java se la conoce como **LookAndFeel** seleccionable. Por ahora el look and feel establece principalmente las características visuales del componente, pero está pensado para añadir también el manejo de características sonoras.

Existen 3 tipos de aspecto o apariencia (look and feel) proporcionados gratuitamente junto al JDK:

- **Apariencia Metal.** Es la apariencia típica de Java, independiente de la plataforma, o multiplataforma . La clase que contiene la información de este look and feel es:
  `javax.swing.plaf.metal.MetalLookAndFeel`
  y el aspecto de una ventana sería el siguiente:
- **Apariencia Motif.** Es la apariencia para plataformas Unix (por ejemplo, para el sistema operativo Linux). La clase que contiene la información de este look an feel es:
  `com.sun.java.swing.plaf.motif.MotifLookAndFeel`
  y el aspecto de una ventana sería el siguiente:
- **Apariencia Windows.** Es la apariencia para plataformas Windows. La clase correspondiente es:
  `com.sun.java.swing.plaf.windows.WindowsLookAndFeel`
  y el aspecto de una ventana sería el siguiente:

Realmente, **con Swing, puede establecerse cualquiera de esas apariencias, con independencia de la plataforma en la que nos encontremos. Para ello usaremos el método `setLookAndFeel()` de la clase `UIManager`, pasándole como argumento la cadena que indica el nombre de la clase que tiene la información de esa apariencia, indicando el paquete en el que está**, tal y como la hemos indicado anteriormente. Veamos como.

7.1. Ejemplos de elección de un LookAndFeel

Ejemplos de elección de un LookAndFeel

Ya hemos visto los aspectos que nos proporciona Java para elegir, las clases que los implementan, e incluso los paquetes que contienen a esas clases.

¿Pero sabemos cómo seleccionar uno de esos aspectos?

Bueno, lo vamos a saber ahora mismo.

- **Para establecer el aspecto multiplataforma de Java, llamado aspecto Metal:**
  `UIManager.setLookAndFeel(javax.swing.plaf.metal.MetalLookAndFeel);`
- **Para establecer el aspecto de plataformas Unix, llamado Motif:**
  `UIManager.setLookAndFeel(com.sun.java.swing.plaf.motif.MotifLookAndFeel);`
- **Para establecer el aspecto de plataformas Windows**
  `UIManager.setLookAndFeel(com.sun.java.swing.plaf.windows.WindowsLookAndFeel);`

También es posible establecer el aspecto multiplataforma y el aspecto del sistema en que se ejecute la aplicación usando métodos específicos que obtienen las clases adecuadas para pasárselas como parámetros al método **`setLookAndFeel()`** :

```
UIManager.setLookAndFeel(UIManager.getCrossPlatformLookAndFeelClassName());
```

El método **`getCrossPlatformLookAndFeelClassName()`** devuelve el nombre de la clase que implementa el aspecto multiplataforma por defecto, el aspecto Metal (o Java Look and Feel). Ese nombre de clase que devuelve incluye el paquete que la contiene, y al pasárselo como parámetro al método **`setLookAndFeel()`**, éste establece el aspecto de la aplicación a Metal.

```
UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
```

En este caso, el método **`setLookAndFeel()`** recibe como parámetro un nombre de clase, que es justamente el que devuelve el método que se le envía como argumento. Ese método es **`getSystemLookAndFeelClassName()`**, y el nombre de clase que devuelve es el de la correspondiente al look and feel del sistema en el que se está

ejecutando la aplicación.

En **AWT** esto no era posible. Los componentes de una aplicación siempre tenían el aspecto de la plataforma en la que se ejecutaban, y eso no casaba muy bien con un lenguaje que presume de ser multiplataforma. Por eso es una de las mejoras que se introdujeron con Swing.

7.2. LookAndFeel para Macintosh

LookAndFeel para Macintosh

¿Sólo existen los aspectos Metal, Motif y Windows?¿No es posible disponer de más aspectos para Java?

Realmente sí. De hecho, hay toda una serie de métodos destinados a permitir la programación de clases para nuevos aspectos, que podrían ser instalados. Lo que ocurre es que esos son los tres únicos aspectos que nos proporciona el lenguaje directamente con el JDK, gratuitamente y disponibles. En la mayoría de los casos deberían ser suficientes.

De hecho, además de los tres aspectos suministrados con el JDK, Sun ha creado **un "look and feel" específico para plataformas Macintosh**, pero este se suministra e instala por separado, no viene incluido en el JDK.

El aspecto de una ventana con apariencia MacOS sería el siguiente:

Puedes ver que es posible cambiar el aspecto de una aplicación incluso de forma dinámica cuando ésta ya está abierta, usando el menú Look & Feel de la aplicación SwingSet Demo que aparece como ejemplo en Java Web Start.

También hemos incluido un menú Look & Feel en la aplicación de la ecuación de segundo grado, que cambia el aspecto de la aplicación, dando a elegir entre dos posibles. Así podrás observar los comentarios del código para ver dónde se produce el cambio.

8. Cerrar la ventana y la aplicación

Cerrar la ventana y la aplicación

*CASO. **Carmen** le explica a **Víctor** que sólo le queda una cosa para continuar por sí solo su preparación como programador en Java con técnicas de programación orientada a objetos y componentes gráficos, se trata de la finalización de las aplicaciones. Dice que es importante terminar bien las aplicaciones y que éstas, normalmente, no finalizan al cerrar las ventanas, ya que pueden quedar muchos recursos en ejecución.*
Para finalizar las aplicaciones es necesario llevar a cabo una serie de operaciones de cierre, imprescindibles para evitar la pérdida de datos, la innecesaria ocupación de memoria y la ejecución descontrolada de tareas.

A todo cerdo le llega su San Martín, dice el refrán, y de igual manera toda aplicación que se ejecuta debe terminar tarde o temprano.

¿Pero cómo se hace? ¿Es lo mismo cerrar la ventana principal de la aplicación que cerrar la aplicación?

En Swing, las ventanas se crean automáticamente con un icono en la esquina superior derecha que cierra la ventana de la aplicación, sin necesidad de que hagamos nada más.

Es cómodo, y es a lo que estamos acostumbrados. Disponer de esa funcionalidad sin necesidad de hacer nada adicional está bien. Pero si por ejemplo queremos incluir una opción del menú para salir de la aplicación, tendrá que hacerlo el programador.

Pero internamente, una cosa es cerrar una ventana, y otra muy distinta es que esa ventana deje de existir completamente, o cerrar la aplicación completamente.

- **Podemos hacer que la 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. De hecho, usamos ese método en los métodos asociados al menú que permite seleccionar el LookAndFeel de nuestras aplicaciones de ejemplo, para que al ocultar y volver a mostrar la aplicación, se dibuje con ese nuevo aspecto.
- **Podemos 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 al método **`show()`**, aunque deberían construir de nuevo toda la ventana.
- **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()`**

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

En el siguiente apartado puedes ver cómo se realizará esta acción.

8.1. Operaciones de cierre por defecto para ventanas (setDefaultCloseOperation)

Operaciones de cierre por defecto para ventanas (setDefaultCloseOperation)

En el apartado anterior hemos hablado de que el método **`setDefaultCloseOperation()`** de la clase **`JFrame`** permite establecer, como su nombre indica, qué tarea por defecto debe realizarse cuando se cierra la ventana.

Esa tarea por defecto puede seleccionarse de entre una lista de cuatro posibles.

¿Cómo se le indica al método la tarea seleccionada?

Como un número entero del 0 al 3, que es el parámetro que admite. Pero ese número entero no se suele dar como tal, sino como una constante de clase, que sirve para "ponerle un nombre significativo" a cada uno de los valores posibles, que nos ayude a recordar su significado.

Los valores que se le pueden pasar como parámetros a este método son una serie de constantes de clase (estáticas).

En este ejemplo, es más fácil recordar lo que hace la llamada al método

```
setDefaultCloseOperation (EXIT_ON_CLOSE);
```

que recordar lo que hace la llamada totalmente equivalente al mismo método

```
setDefaultCloseOperation (3);
```

Como decíamos, las constantes de clase que se le pueden pasar como parámetros al método son:

- **`DO_NOTHING_ON_CLOSE`** (definida en **`WindowConstants`** con el valor 0): **No hace nada**; requiere que el programa maneje la operación en el método **`windowClosing()`** de un objeto **`WindowListener`** registrado para la ventana.
- **`HIDE_ON_CLOSE`** (definida en **`WindowConstants`** con el valor 1): **Oculta automáticamente el marco o ventana** después de invocar cualquier objeto **`WindowListener`** registrado.
- **`DISPOSE_ON_CLOSE`** (definida en **`WindowConstants`** con el valor 2): **Oculta y termina (destruye) automáticamente el marco o ventana** después de invocar cualquier objeto **`WindowListener`** registrado.
- **`EXIT_ON_CLOSE`** (definida en **`JFrame`** con el valor 3): Sale de la aplicación usando el método **`System.exit()`**.Al estar definida en **`JFrame`**, sólo se puede usar con aplicaciones, no con applet.

El valor se fija por defecto a **`HIDE_ON_CLOSE`**. (Eso al menos es lo que dice la API de Java, pero por ejemplo, NetBeans lo fija por defecto a **`EXIT_ON_CLOSE`**)

8.2. Cierre de la aplicación

Cierre de la aplicación

Todas estas posibilidades sólo actúan sobre la ventana, pero si se quiere cerrar la aplicación, habrá que añadirle un Listener (un escuchador de eventos) para decirle de forma explícita que al cerrar la ventana, también cierre la aplicación ejecutando en el manejador de ese evento el método **`System.exit(0);`**

- Una forma sencilla de hacerlo es añadir ese escuchador mediante el uso de una clase Adapter (concretamente **`WindowAdapter`**)
- WindowAdapter nos proporciona una implementación del interface **`WindowListener`**, aunque con métodos vacíos, cuyo cuerpo no contiene ninguna sentencia.
- El interface **`WindowListener`** establece los métodos que debe incluir cualquier escuchador de eventos de ventana, y serán básicamente uno para cada evento posible. La lista de eventos posibles son:
  - abrir la ventana (**`windowOpened`**). Evento que se produce cuando la ventana se hace visible por primera vez, es decir, cuando se abre la ventana.
  - activar la ventana. (**`windowActivated`**). Evento que se produce cuando la ventana pasa a ser la ventana activa.
  - desactivar la ventana (**`windowDeactivated`**). Evento que se produce cuando la ventana deja de ser la ventana activa.
  - iconificar la ventana (**`windowIconified`**) Evento que se produce cuando la ventana pasa a estar minimizada.
  - deiconificar la ventana (**`windowDeiconified`**). Evento que se produce cuando la ventana deja de estar minimizada.
  - cerrar la ventana (**`windowClosing`**). Evento que se produce cuando la ventana ha sido cerrada por el usuario desde el menú.
  - ventana cerrada (**`windowClosed`**). Evento que se produce cuando la ventana ha sido cerrada usando **`dispose()`**
- Esto nos permite redefinir sólo el método correspondiente al evento que nos interesa capturar, y no todos los que establece el **`interfaceWindowListener`**, que son siete en total. Y de todos ellos a nosotros sólo nos interesa decir qué es lo que habrá que hacer cuando se presente uno concreto, que es que la ventana se cierre (**`windowClosed`**). De todos los posibles, sólo nos interesa indicarle qué hay que hacer cuando se cierra la ventana:
  - Al cerrar la ventana hay que ejecutar el método **`System.exit(0)`** que cierra totalmente la aplicación, liberando todos los recursos que ocupara. El argumento entero de **`exit()`** se usa como un código de estado. Por convenio, se considera que un valor distinto de cero se usa para indicar que la aplicación terminó anormalmente, y un valor cero para indicar que se ha cerrado la aplicación porque es lo que quería hacerse, es decir, se ha cerrado normalmente.

En el siguiente cuadro, tienes un esquema del código necesario tanto para establecer la operación por defecto al cerrar la ventana, como el código necesario para cerrar la aplicación:

```
      //...
Jframe f= new JFrame("Ejemplo de cierre de unaAplicación");
//...
//Código donde se hacen operaciones con esa ventana que acabamos de crear
f.setBounds(100,100,300,300);
f.setVisible(true);
/* Definimos la operación que queremos hacer por defecto, cuando se cierre la ventana.
*  En este caso, destruir toda la memoria que ocupara la ventana.*/
f.setDefaultCloseOperation(DISPOSE_ON_CLOSE);
//..
/* Definimos un listener que escucha el evento de cerrar la ventana, y cuando se produce
*  lo captura y ejecuta el método exit(0) de la clase System, que sale de la aplicación,
*  cerrándola, deteniendo su ejecución.
*/
f.addWindowListener(new WindowAdapter(){
      public void windowClosed(WindowEvent e){
           System.exit(0);
      }
});
```

En el ejemplo de la aplicación de la ecuación de segundo grado también puedes observar comentadas las instrucciones que cierran la aplicación.

Autoevaluación