DESARROLLO DE INTERFACES

TÉCNICO EN DESARROLLO DE APLICACIONES MULTIPLATAFORMA

Introducción a la confección de interfaces

Un **lenguaje de programación** consiste en un conjunto de reglas y normas que permiten a una persona (en este caso un programador) escribir un conjunto de instrucciones interpretables por un ordenador, cuyo **objetivo** es **controlar** diferentes **comportamientos lógicos o físicos de una máquina**.

De manera tradicional, se ha establecido una clasificación entre lenguajes de bajo y alto nivel. Los primeros se encuentran más cerca de lo que es capaz de entender un ordenador, ejercen un control directo sobre el hardware y están más alejados de la lógica humana (lenguaje máquina con 0 y 1 o lenguaje ensamblador).

Los anteriores resultan muy difíciles de entender por una persona. Por esa razón, aparecen los lenguajes de alto nivel, los cuales pueden ser descritos utilizando reglas comprensibles por el programador, con un lenguaje más cercano al natural. Será durante el proceso de compilación del código fuente de los programas, cuando estos se traduzcan a un lenguaje de bajo nivel, capaz de ser entendido por una máquina.

Ahora bien, las herramientas desarrolladas a través de un lenguaje de programación, sea del tipo que sea, **requieren del desarrollo de una interfaz** que permita una interacción con el usuario de la misma, de lo contrario, se requeriría que todos fuéramos programadores expertos para poder utilizar cualquier aplicación atendiendo a su lenguaje fuente.

Paradigmas de programación

Un paradigma de programación define un **estilo de programación**. Los paradigmas describen la **estructura del programa** que va a dar solución a los problemas computacionales.

En primer lugar, encontramos el **modelo imperativo**, que consiste en un conjunto de instrucciones ordenadas de **forma secuencial y claramente definidas** para su ejecución en una máquina, es decir, definen un paso a paso. Este modelo se divide en **otros tipos**:

- **Programación estructurada:** Incluye **estructuras de control** que permiten evaluar los casos para decidir entre un camino de instrucciones u otro. También se incorporan estructuras **iterativas**.
- **Programación procedimental o basada en funciones:** Subdivide en **subrutinas** y funciones de menor tamaño que **simplifican** la programación, aligerando su implementación y posterior mantenimiento.
- **Programación modular:** Finalmente, este tipo de programación permite desarrollar cada **programa** de forma completamente **independiente** al resto del código, lo que agiliza las tareas de implementación y prueba. Será en la parte final del proceso cuando **se combinen todos** los módulos, creando el software definitivo.

Algunos lenguajes conocidos que utilizan la **programación imperativa** son: Java, C, C#, Python o Ruby.

En el modelo imperativo se indica la **secuencia de pasos exacta** que se ha de seguir para resolver un problema.

Por el contrario, en el caso del **modelo declarativo**, **no se describen los pasos**, sino el **problema** que se plantea.
Algunos ejemplos de lenguajes descriptivos son HTML, CSS y SQL.

Programación orientada a objetos, eventos y componentes

Encontramos otros modelos de programación que incluyen características propias de los definidos en el apartado anterior. Es el caso de la programación orientada a objetos, eventos o componentes.

La combinación de estos tres tipos resulta **clave para el desarrollo de interfaces** que veremos en este módulo.

- **Modelo orientado a objetos:** El funcionamiento de este tipo de programas se basa en la creación de **entidades**, que reciben el nombre de objetos, las cuales tienen asociados **atributos, propiedades y métodos**. La interacción entre los objetos permite resolver los problemas de computación planteados. Algunos lenguajes de programación orientados a objetos son: Java, Ruby, Visual Basic, Perl, PHP o Python, entre otros.
- **Modelo basado en eventos:** Este modelo es uno de los más recientes. Su funcionamiento viene determinado por **acciones** externas, por ejemplo, la pulsación sobre un botón. Uno de los lenguajes típicos de este tipo de programación es **JavaScript**, que utiliza manejadores de eventos, tanto en el lado del cliente como del servidor (Node.js).
- **Modelo basado en componentes:** La clave de este último modelo es la **reutilización de módulos** de software desarrollados previamente.

Para llevar a cabo esta tarea la mayoría de los entornos de desarrollo integrado (IDE) permiten desarrollar componentes visuales, permitiendo empaquetar el código para reutilizarlo posteriormente.

Herramientas propietarias y libres de edición de interfaces

La motivación principal para utilizar herramientas de desarrollo software basado en componentes visuales radica en que, una vez empaquetados, se podrán compartir con otros desarrolladores y, por tanto, serán reutilizables. Esto supone un mayor ciclo de vida que el desarrollo tradicional por comandos. Si no se desarrolla utilizando componentes y se realiza de manera directa, se producirá un incremento de tiempo y costes asociados al proyecto.

Destacamos a continuación las principales herramientas de desarrollo software:

NOMBRE	LICENCIA	LENGUAJES SOPORTADOS	ENLACE
Visual Studio *Community	Propietaria *Libre	C#, HTML, Javascript, XML	https://visualstudio.microsoft.com/es/
Mono Develop	Libre	C#, Java, .NET, Python	https://www.monodevelop.com
Glade	Libre	C++, C#, Java, Python	https://glade.gnome.org
NetBeans	Libre	Java, HTML, PHP, Python	https://netbeans.org
Eclipse	Libre	Java, C++, PHP	https://www.eclipse.org

Visual Studio

Entre las fortalezas más importantes de este IDE se encuentra el **uso de lenguajes multiplataforma**. Se puede escribir en los lenguajes C#, F#, Razor, HTML5, CSS, JavaScript, Typescript, XAML y XML.

Además, este entorno de desarrollo incorpora la funcionalidad de autocompletado de código mientras estamos programando, por lo que es fácil detectar los problemas en **tiempo real**, ya que nos recuerdan que es posible que se esté cometiendo algún fallo a través de líneas rojas y onduladas que se muestran en el editor bajo una línea o fragmento de código. A la hora de depurar el código, permite ir paso a paso, estableciendo puntos de interrupción, por procedimientos y por instrucciones.

Por último, cabe destacar que permite **administrar el código en los repositorios más utilizados** en la actualidad como son GIT, GitHub y Azure DevOps. De esta forma, es posible controlar las modificaciones entre las diferentes versiones de nuestros proyectos y poder realizar copias de seguridad de los archivos durante el desarrollo del mismo.

MonoDevelop

Este IDE libre y gratuito proporciona las funcionalidades propias de un **editor de texto**, además de las propias de un entorno para **depurar y gestionar proyectos.**

Entre sus principales ventajas encontramos que permite trabajar con algunos de los lenguajes más demandados en la actualidad, como son C#, Java, .NET y Python. Pertenece a **Unity**, motor de videojuegos multiplataforma por excelencia. Esta plataforma es muy interesante porque permite desarrollar tanto para Windows, Mac Os X y Linux.

Glade

Este programa ayuda a la creación de interfaces gráficas de usuario y es muy utilizada en entornos **XML**. También permite el **desarrollo de interfaces gráficas basadas en lenguaje C, C++, C#, Java, Python…**

Dado que la interfaz es bastante intuitiva, se puede aprender rápido y obtener un dominio de la herramienta invirtiendo poco tiempo. La principal diferencia respecto a las demás propuestas es que está diseñada pensando especialmente en **GNU/Linux**.

NetBeans

NetBeans es una **herramienta gratuita y de código abierto.** Al igual que Eclipse, que veremos en el siguiente apartado, es uno de los entornos de desarrollo más utilizados para el desarrollo de interfaces a través lenguaje Java, aunque también permite utilizar otros lenguajes como PHP o Python.

Este IDE permite extender el entorno con un gran número de módulos que agrupan clases de Java y que permiten interactuar con las APIs de NetBeans.

Eclipse

Este IDE es de **código abierto y multiplataforma**. Dispone de la funcionalidad Graphical Layout, que nos permite visualizar el contenido en vista de diseño y desarrollar componentes visuales de una forma rápida e intuitiva. Cabe destacar su componente 'Palette', un panel que permite crear botones, cuadros de texto, cuadrículas, insertar imágenes, etc.

Para completar esta asignatura vamos a utilizar el entorno de desarrollo Eclipse, puesto que, en la actualidad, su uso es cada vez más frecuente en lo que respecta al desarrollo de interfaces de forma profesional.

Para poder obtenerlo, desde el sitio web de Eclipse primero se selecciona la versión que corresponda con el equipo con el que estemos trabajando, se descarga y, a continuación, se instala a través de unos sencillos pasos. Este proceso ocupa pocos minutos. Ahora bien, también se debe tener instalado Java Development Kit (JDK) correspondiente al sistema operativo del equipo, cuya descarga puede realizarse desde la página web de Oracle.

Una vez completado este proceso, ya tendríamos instalado todo el entorno básico para el desarrollo de interfaces posterior. Para ejecutar Eclipse basta con pulsar sobre el icono de la aplicación, normalmente con el nombre de **Eclipse Installer.** Finalmente, selecciona 'Eclipse IDE for Enterprise Java Developers', luego pulsa 'Install' y posteriormente 'Launch'.

Librerías
AWT y Swing

Algunos lenguajes de programación (entre ellos Java) utilizan librerías, un **conjunto de clases** con sus propios atributos y métodos ya implementados. De esta forma, pueden utilizarse posteriormente para cualquier desarrollo reutilizando su código, lo cual **reduce considerablemente el tiempo de programación**. En cuanto al desarrollo de interfaces gráficas, para poder implementarlas debemos usar librerías que permiten el desarrollo de las mismas. En Java se distingue entre la librería AWT y Swing.

Para poder utilizar los métodos y atributos de estas clases, es necesario **importar** las librerías en Java. Para ello, se utiliza la palabra clave import, seguida del nombre de la librería requerida, en concreto, de la ruta del paquete que se va a agregar. Esta importación se realiza justo después de la declaración del paquete, si esta existe.

```
import javax.swing.*;
import java.awt.*;
```

AWT (Abstract Window Toolkit) se desarrolló en primer lugar. Esta librería permite la creación de interfaces gráficas a través de la importación del paquete **java.awt.** Dos de sus funcionalidades más importantes son el uso de la clase 'Component' y el de la clase 'Container'. La primera define los controles principales que se sitúan dentro del elemento **container o contenedor** (este último hace referencia a la **pantalla** en la que se muestra la interfaz de aplicación a desarrollar).

	AWT	SWING
Usa componentes del S.O.	sí	no
Dibuja sus propios componentes	no	sí
El S.O. maneja los eventos	sí	no
Java maneja los eventos	no	sí
La apariencia cambia con el S.O.	sí	no
La apariencia es estática	sí	no
Se pueden personalizar	no	sí

En la actualidad, la librería **Swing** supone la **evolución** de la anterior, eliminando algunas limitaciones que presentaba AWT (como el uso de barras de desplazamiento). Swing incorpora múltiples herramientas, métodos y componentes que permiten diseñar cualquier tipo de interfaz. A través de un entorno gráfico, permite crear un diseño desde cero arrastrando los componentes hasta la paleta de diseño, mientras que al mismo tiempo se va generando el código asociado. Conocer el funcionamiento de ambas **vistas, diseño y código**, permite **adaptar el funcionamiento a las especificaciones** de aplicación buscadas.

Instalación de Swing en Eclipse

Para la implementación de interfaces gráficas en Java, se va a utilizar la librería Swing, un kit de herramientas que permite desarrollar interfaces gráficas de usuario para programar Java. Esta librería, a diferencia de su antecesora, **garantiza** que el **diseño y comportamiento** de las aplicaciones será exactamente **el mismo**, **independientemente** del **sistema operativo**. Esto se debe a que **AWT** utiliza los **controles nativos** del sistema operativo en el que se encuentra.

A diferencia de la librería AWT, Swing proporciona una apariencia que **puede emular** varias plataformas y utilizar **componentes visuales más avanzados**. El proceso de instalación y configuración de esta librería en el entorno de desarrollo de Eclipse se describe a continuación:

- **Desde el menú Help seleccionamos Install New Software.**
- **En el menú desplegable de la parte superior de la nueva pantalla se selecciona la versión del entorno que estemos utilizando.**

En ese momento aparecerán todos los paquetes disponibles para esa versión y seleccionaremos todos los paquetes que comienzan por **SWT y WindowBuilder.** Recuerda que se encuentran dentro de la carpeta **General Purpose Tools.** Después, pulsamos continuar.

Primera clase con Java Swing. JFrame

La importación de la librería Swing de Java se realiza usando la sentencia, **import javax.swing**. De esta forma, todas las clases contenidas en el paquete serán importadas. En el caso de querer importar solo una de ellas, basta con indicarlo tras la palabra swing. Por ejemplo, si se va a utilizar la clase que crea los botones, se usaría **import javax.swing.JButton.**

Una de las clases más importantes del paquete Swing es **JFrame**, puesto que se encarga de crear las **ventanas** sobre las que se añaden el resto de elementos. La llamada a esta clase conllevará la **aparición de la vista de diseño** (Design).

En algunas ocasiones, se puede confundir la clase JFrame con la clase **JPanel**, pero mientras que la primera define una ventana completa, la segunda es solo un **contenedor de componentes**, por lo que **dentro de un Jframe podríamos encontrar múltiples JPanel (en android es el equivalente a Layout)**.

Se puede crear una clase Jframe de la manera habitual, creando una clase en Java y, a continuación, importando manualmente el paquete Swing y codificando los métodos relativos a JFrame y el resto de componentes. El resultado sería exactamente el mismo, pero se recomienda realizarlo usando la creación de clase con JFrame que se describe a continuación, puesto que, de lo contrario, no se tendría acceso a la vista de diseño que se verá en el apartado 9. La **creación** de nuestra primera **clase JFrame** se realiza en dos sencillos pasos:

- Desde **File** o pulsando con el botón derecho sobre el nombre del proyecto que se está desarrollando, seleccionamos **New.**

  Se despliega un menú con diferentes opciones, habitualmente las que se han usado más frecuentemente. Si no aparece JFrame, seleccionaremos **Other.**

• La clase JFrame se encuentra dentro de las carpetas **WindowBuilder y Swing Designer**. Seleccionamos **JFrame** y pulsamos **Next.** Finalmente, se solicita el nombre y se pulsa Finish.

Análisis del entorno de diseño en Eclipse

El área de diseño para desarrolladores de Java en Eclipse permite **desarrollar interfaces añadiendo componentes gráficos directamente**, sin necesidad de programar líneas de código. Para ello, es importante conocer la distribución de la interfaz de trabajo de este IDE.

La vista de diseño nos permite insertar cualquier tipo de elemento en la interfaz sin necesidad de escribir el código, ya que este se genera automáticamente y se puede consultar desde la pestaña 'Source'. Es decir, desde 'Design' es posible añadir todos los elementos que se quieran incluir en la aplicación. Así definiremos la parte visual y, desde el apartado dedicado al código, se cargarán los métodos relativos a cada uno de los objetos insertados, sobre los cuales podremos incluir el resto del código necesario para definir su comportamiento exacto.

A continuación, se describen los diferentes **grupos de herramientas** que podemos encontrar en Eclipse, tanto los de tipo general, necesarios para la programación de cualquier aplicación con Java, como los específicos del área de diseño.

Toolbar

En la barra de herramientas o 'Toolbar' de Eclipse se encuentran los iconos relativos a las acciones genéricas de programación, como la creación de paquetes, clases... Uno de los botones más importantes es el encargado de la ejecución del programa. Al hacer clic sobre la flecha que se encuentra a su derecha, se podrá seleccionar cuál de entre las clases Java del proyecto actual se quiere ejecutar. En el desplegable que aparece como "Run As" se podrá seleccionar el tipo de ejecución como Aplicación de Java.

Vista de diseño
General

La zona de diseño es la ventana principal del entorno de desarrollo con **WindowBuilder**, puesto que es la zona en la que se colocan los elementos de la interfaz y, además, donde se encuentran todos los componentes y características de diseño necesarios para el desarrollo de una interfaz gráfica. En la zona de la derecha de la pantalla se muestra una previsualización de la aplicación que se está implementando, podríamos decir que es el lienzo sobre el qué dibujar la interfaz.

Vista de diseño
Palette

En la parte de la izquierda del área de diseño aparece el menú **Palette**, que recoge todos los componentes, propiedades y contenedores que se utilizan en la creación de una interfaz gráfica. Desde el menú Palette se realiza todo el diseño de la interfaz, ya que incorpora múltiples herramientas como los **containers,** que definen el tipo de contenedor donde se ubican los componentes; **layouts**, que determinan la distribución exacta de los elementos; o los componentes (**components**) que queramos utilizar en la interfaz (etiquetas, campos de texto o botones, entre otros).

Los componentes gráficos son los elementos que permiten al usuario interaccionar con la aplicación. Cada uno de los componentes corresponde con una clase en Java, es decir, cada componente tendrá sus propios métodos y atributos, por lo que, cuando se quieran utilizar, bastará con crear una instancia del objeto deseado.
Para insertarlos en la zona de diseño basta con hacer clic sobre el componente y arrastrarlo hasta la zona del contenedor exacta en la que se va a ubicar.

Vista de diseño
Structure

La última sección de la vista de diseño en Eclipse recibe el nombre de **Structure** y está formada por dos partes claramente diferenciadas: components y propierties.

- **Components:** Esta zona muestra un **resumen** de todos los componentes que se han colocado en el diseño de la interfaz, como si de un **explorador de carpetas** y archivos se tratase, pero en este caso nos muestra los elementos de diseño. Como se puede observar, aparece el **nombre de la instancia** del objeto que se ha creado en el caso de los botones btnNewButton y btnNewButton_1, pero este es solo el nombre de la variable en Java; el **texto** que **se muestra al usuario** puede ser diferente y, en la mayor parte de los casos, lo será. Este se muestra **entre comillas**.
- **Propierties:** Cada uno de los componentes dispone de diferentes propiedades que podrán ser modificadas desde la ventana que se encuentra a la izquierda de la paleta de elementos (Palette).

  Una de las propiedades más importantes es la llamada **Variable**, que recoge el **nombre de la instancia** del componente creado en código Java. Por ejemplo, si se inserta un componente de tipo botón (JButton), el nombre que recibe la nueva instancia del objeto sería btnNewButton. Si se incluye otro botón, este nombre será diferente: no pueden existir dos elementos con el mismo nombre.

  En el caso de los botones, otras de las propiedades de aspecto que aparecen y que permiten definir la apariencia son: ‘background’, para seleccionar el color de fondo del botón; ‘enabled,’ que permite habilitar o deshabilitar su funcionalidad; ‘font’ y ‘foreground’, que definen el tipo de letra, el tamaño y el color, entre otras.

Creación de un JFrame

En el siguiente código se muestran cada uno de los pasos descritos en el planteamiento mediante el uso de los métodos setSize, setVisible y setDefaultCloseOperation, y se definen todos los parámetros de diseño necesarios para generar una ventana desde cero.

```
import javax.swing.*;

public class MiPrimeraVentana {

	public static void main(String[] args) {

	JFrame f = new JFrame("Mi primera ventana");

	f.setSize(400, 400);
	f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
	f.setVisible(true);
	}
}
```

El resultado del código anterior es una ventana como la que se muestra en la siguiente imagen, en la que se puede apreciar las dimensiones establecidas de 400 x 400 píxeles de ancho por y alto y el nombre de la ventana.
Es importante destacar que unas de las principales diferencias a la hora de crear un JFrame directamente con el código es que no estará habilitado el modo Design.

Creación de un botón

Para crear un JFrame seleccionamos en el menú File -> New -> Other -> JFrame. Como podemos ver, esta clase crea el siguiente código implementado en la vista 'Source'. Además, por defecto, se importa la librería AWT:

Añadir objetos de tipo JButton: Desde la vista Design añadimos un componente JButton y modificamos su propiedad 'text' con el contenido «Aceptar». Repetimos el proceso para el botón «Cancelar» y lo colocamos en otro de los 'containers'.

¿Qué es GitHub?

Es un repositorio de desarrollo software que utiliza un sistema de control de versiones. Es muy útil en proyectos que tienen una gran calidad de archivos de código.

¿Qué ventajas tiene usar GitHub?

Permite el **trabajo colaborativo** entre desarrolladores, comparar versiones de un proyecto, crear copias de seguridad, mostrar gráficas sobre el flujo de trabajo de cada programador e incluso como si fuera una red social, realizar suscripciones y tener seguidores.

En la actualidad se ha convertido en una herramienta muy importante tanto desde el punto de vista educativo como profesional, dado que permite trabajar en equipo sin estar de manera presencial en un mismo sitio.