MEDAC
Instituto Oficial de Formación Profesional

DESARROLLO DE INTERFACES
TÉCNICO EN DESARROLLO DE APLICACIONES MULTIPLATAFORMA

Creación de componentes

04

ÍNDICE

OBJETIVOS

Identificar las herramientas para el diseño y prueba de componentes.

Crear componentes visuales.

Definir sus propiedades y asignar valores por defecto.

Documentar los componentes creados.

/ 1. Introducción y contextualización práctica

El **desarrollo de interfaces gráficas** no solo permite la inclusión de componentes ya existentes, sino que nos permite la creación de nuevos elementos que se ajusten a las características de nuestro proyecto. Habitualmente, se utilizan los componentes recogidos en las distintas librerías gráficas, y gracias a la herencia se pueden tomar como base para el desarrollo de futuros componentes.

Todo componente supone un elemento **JavaBean**, el cual presenta ciertas características que definen su funcionamiento, como son la introspección y la reflexión, así como la posibilidad de customización y la generación de una comunicación activa entre el usuario y la interfaz a través de los eventos.

La integración de cualquier componente se basa en la **generación de un archivo Jar** que puede ser importado en cualquier entorno de desarrollo, permitiendo así el uso de sus componentes.

En cuanto al diseño gráfico, existe una **librería denominada Graphics**, a través de la cual se podrán realizar múltiples diseños gráficos sobre ventanas y paneles.

La combinación entre componentes, ya sean los incorporados en las librerías de desarrollo gráfico, en la creación de nuevos elementos, o en el uso de librerías gráficas, permiten diseñar multitud de escenarios que se adaptan a cada casuística concreta.

Los componentes presentan unas características y propiedades ya definidas, pero es posible crear nuevos utilizando los ya existentes, modificando y adaptando aquellos elementos necesarios.

A continuación, vamos a plantear un caso a través del cual podremos aproximarnos de forma práctica a la teoría de este tema.

Fig. 1. Componentes visuales más utilizados de la librería Swing.

Audio intro. "Creación y utilización de componentes visuales en interfaces"
https://bit.ly/2DhGgVK

/ 2. Desarrollo de software en componentes. Reutilización de software

Crear un componente consiste en crear un bloque de *software* reutilizable, es decir, se va a implementar **un elemento** a través de un determinado lenguaje de programación que va a **poder ser utilizado en cualquier otro proyecto**. Los componentes que se vieron en el tema 2 son bloques de código reutilizable, es decir, los componentes de la librería Swing son módulos ya desarrollados que pueden ser utilizados en cualquier proyecto que requiera de diseño gráfico.

Cuando hablamos de **reutilización del código**, nos referimos a la **reutilización de librerías, frameworks o kit de herramientas.**

Para el desarrollo de interfaces gráficas visto hasta ahora con *Swing*, se han tomado algunos de los componentes ya implementados y se han personalizado modificando algunas de sus propiedades. Ahora bien, utilizando como base estos elementos ya desarrollados se podrán crear otros nuevos que presenten otras características que el componente tomado como referencia no incorpora. El desarrollo de cero de un componente requeriría de líneas y líneas de código. En la actualidad, se toman como base los ya implementados y se les incorporan ciertas mejoras.

Son múltiples los beneficios que supone la **reutilización de módulos software:**

- Debido a la disminución del tiempo necesario para el desarrollo **se reducen los costes del proyecto.** Normalmente, el coste total de un desarrollo está estrechamente ligado al tiempo necesario para su creación.
- Las **pruebas de software se simplifican**, puesto que los módulos reutilizados ya se habrán probado previamente. De esta forma, solo será necesario someter a pruebas a los nuevos desarrollos.
- **Mejora la calidad del software.** Este beneficio es uno de los más importantes, puesto que en cada nueva implementación, un módulo ya creado podrá ser adaptado a nuevas casuísticas que se incorporarán a las anteriores.

Podemos hablar de la reutilización de *software* a tres niveles: en primer lugar, a nivel de clase cuando hablamos de clases y algoritmos, en segundo lugar, a nivel de diseño a través de los patrones de diseño, y finalmente, a nivel de arquitectura.

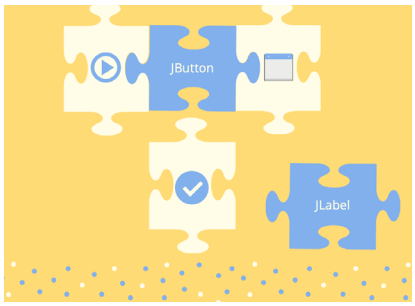

Fig. 2. Ilustración de reutilización de código y componentes.

/ 3. Concepto de componentes y características

Un **componente** es un módulo de código ya implementado y reutilizable que puede interactuar con otros componentes *software* a través de las interfaces de comunicación.

Los componentes permiten la implementación de sistemas utilizando componentes ya desarrollados, y por tanto, probados, lo que conlleva a una notable reducción del tiempo de implementación y los costes asociados.

Para conseguir una reutilización eficiente del *software* es necesario que esta esté definida de la manera más generalizada posible, puesto que a partir de los componentes más generales se podrían implementar múltiples versiones modificadas.

Atendiendo al diseño gráfico con *JSwing*, cuando hablamos de un componente, nos referimos a alguno de los elementos que se sitúan en la ventana, directamente sobre el *JFrame* y *JDialog*, o sobre un JPanel, y que le aporta funcionalidad a la interfaz, o, de lo contrario, solo tendríamos una ventana abierta sin más.

Para diseñar un buen componente, es deseable que presente **ciertas características:**

- **La implementación** puede estar realizada con cualquier lenguaje de programación, pero ha de estar completa.
- **Constituye un módulo reutilizable**, ya compilado.
- **Su distribución** se realiza en un solo paquete ejecutable.

La metodología de programación basada en el uso de elementos reutilizables recibe el nombre de **Programación Basada en Componentes** (POC).

Todos los tipos de componentes visuales que presenta la librería JSwing se muestran a continuación, a modo de resumen. Se podrá tomar cualquiera de ellos para desarrollar uno nuevo desde cero.

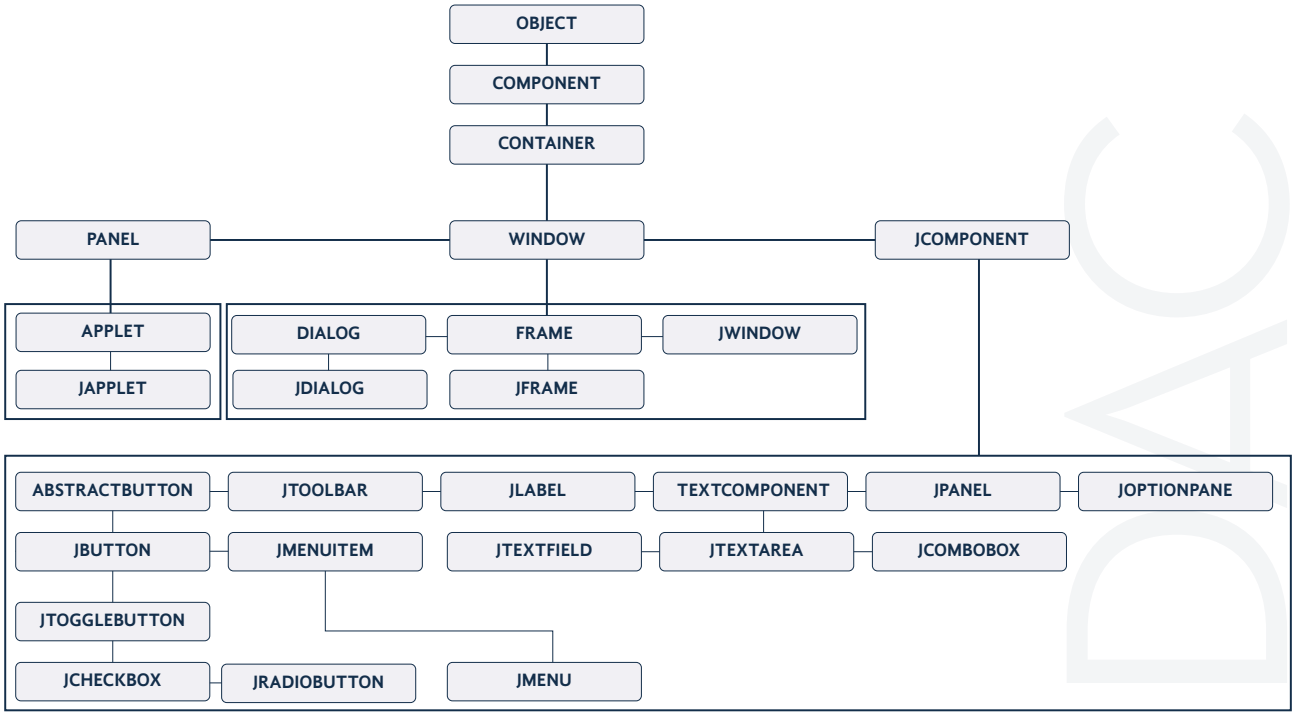

Fig. 3. Esquema tipos de componentes de la librería Swing.

/ 4. Propiedades y atributos

Las **propiedades de un componente** definen los datos públicos que forman la apariencia y comportamiento del objeto. Las propiedades pueden modificar su valor a través de los métodos que definen el comportamiento de un componente.

Por ejemplo, si hablamos de un componente tipo *JButton*, una de sus propiedades será *font*, la fuente del texto que aparece dentro del elemento. Este valor será consultado o modificado a través de métodos tales como **setFont (...).**

Los métodos clave que permiten analizar el contenido de una propiedad o atributo son los de tipo **get**, mientras que para modificar su valor se utilizan los métodos **set**.

Ámbito público	Ámbito privado	Ámbito estático
Una propiedad de ámbito público puede ser utilizada desde cualquier parte de la aplicación.	Una propiedad de tipo privada solo es accesible desde la clase donde se ha creado.	Pueden ser utilizadas sin la necesidad de crear una instancia del objeto al que está referida.

Tabla 1. Tipos de ámbitos de propiedades.

Se distinguen principalmente dos tipos de propiedades: simples e indexadas.

- **Las propiedades simples** son aquellas que representan solo un valor. Es el caso de los atributos sencillos como los de tipo string, int o boolean, entre otros.

```
JCheckBox checkBox = new JCheckBox("OK");
checkBox.setSelected(true);
```

Código 1. Ejemplo de propiedad simple.

- **Las propiedades indexadas** son aquellas que representan un conjunto de valores en forma de array. Por ejemplo, cuando hablamos de un elemento tipo ComboBox, el listado de valores que se muestran en el menú se recoge dentro de un array, por lo tanto, esta propiedad sería de tipo indexada.

```
JComboBox comboBox = new JComboBox();
comboBox.setModel(new
DefaultComboBoxModel(new String[]
{"1", "2", "3"}));
```

Código 2. Ejemplo de propiedad indexada.

Los **atributos**, aunque son similares a las propiedades, se utilizan para almacenar los datos internos y de uso privado de una clase u objeto.

/ 5. JavaBean. Características

Para el desarrollo de interfaces a través del lenguaje de programación Java, los componentes *software* que permiten su reutilización reciben el nombre de **JavaBeans**. Estos pueden ser representados gráficamente o, por el contrario, no presentar ninguna interfaz visual. Prácticamente, todos los componentes y elementos de la paleta gráfica vista en el entorno de diseño son módulos de tipo JavaBean.

JavaBean es el primer modelo para el desarrollo de componentes de Java e implementa el modelo Propiedad-Evento-Método, es decir, podemos decir que sus características distintivas son: los métodos, las propiedades y los eventos.

Los componentes desarrollados como JavaBean presentan un conjunto de características comunes:

- **Introspection.** La herramienta de desarrollo podrá analizar en profundidad el funcionamiento concreto del JavaBean. La clase BeanInfo ofrece un soporte en el que se recoge información sobre características adicionales.

- **Persistence.** Cualquier nuevo componente JavaBean podrá ser almacenado para ser utilizado posteriormente en cualquier proyecto. La persistencia se logra gracias a la serialización del componente (implements Serializable).

- **Customization.** Será posible modificar casi cualquier propiedad y comportamiento del componente durante su implementación. Para llevar a cabo el nuevo diseño, será posible utilizar código o las propiedades de la vista de diseño.

- **Events.** El componente podrá tener asociadas unas acciones como respuesta a un estímulo. Se trata de una de las características más relevantes de los JavaBean, los eventos o sucesos, a través de los cuales es posible la comunicación entre componentes desarrollados de forma completamente independiente.

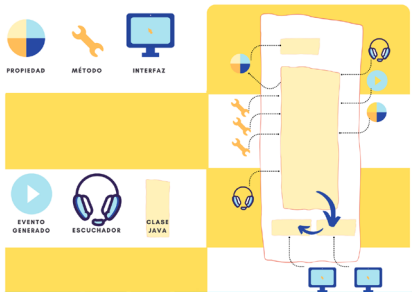

Fig. 4. Estructura de un componente JavaBean.

Audio 1. "Diferencia entre un JavaBean y las clases de Java"
https://bit.ly/304lOkg

/ 6. Creación de un nuevo componente JavaBean

Dado que la creación de un nuevo componente desde cero requiere de muchas líneas de código, lo habitual es utilizar otros componentes ya desarrollados e introducirles mejoras y otros cambios que se adapten a nuevas casuísticas. En este apartado se detalla el proceso completo para la creación de un componente de esta forma.

En primer lugar, es necesario crear un nuevo proyecto para la creación de un nuevo componente. A continuación, para tomar como referencia un componente ya creado, utilizaremos la herencia, es decir, el nuevo extenderá del base. Para implementar la herencia, como seguro que ya conoces de la asignatura de Programación, utilizaremos la palabra reservada extends, seguida del nombre de la clase de la que se hereda.

```
public  class NuevoComponente extends ComponenteBase{};
```

Código 3. Extensión con herencia.

Para la creación de un nuevo componente, la clase que lo implementa ha de ser de tipo JavaBean, por lo que se tienen que cumplir dos características básicas:

1. iene que implementar **Serializable:**

```
public  class NuevoComponente extends ComponenteBase implements
Serializable {}
```

Código 4. Implementación Serializable.

2. Ha de tener un constructor sin parámetros:

```
public  class NuevoComponente extends ComponenteBase implements
Serializable {
        public NuevoComponente(){}
}
```

Código 5. Constructor sin parámetros.

Para definir el nuevo comportamiento, será necesario implementar la nueva funcionalidad, es decir, los métodos.

Es habitual utilizar alguno de los ya existentes y sobrescribir su contenido añadiendo nuevo código. Para redefinir el comportamiento, se utiliza la palabra reservada **@Override**, seguido del mismo nombre del método en la clase de referencia.

```
@Override
protected  void nuevoMetodo ( ... )
{
super.nuevoMetodo( ... );
}
```

Código 6. Sobrescribir un método.

Recuerda que si quieres mantener la misma funcionalidad que el método del componente base, se utiliza el constructor de la clase padre de la que se hereda (**super**) y, a continuación, se añade la nueva implementación.

/ 7. Extender la apariencia y comportamiento de los controles en modo diseño

Al igual que ocurría con los componentes ya creados, el nuevo componente deberá incorporar aquellas propiedades que el que se ha tomado como base no tiene. Es decir, si se va a implementar **un nuevo botón**, habrá que crear solo las propiedades, pero no habrá que implementar las que ya posee el botón base, como pueden ser el tipo de texto o el color de la fuente.

La creación de estas propiedades se realiza de la forma habitual en la que se implementan los atributos de cualquier objeto en Java, s**e añade en la parte superior de la clase.**

```
public   class  NuevoComponente  extends  ComponenteBase  implements
Serializable {
        private tipo(int, String,...) nombrePropiedad;
        public NuevoComponente(){}
}
```

Código 7. Creación de propiedades.

A continuación, se **incorporan los métodos set y get** a la clase. Estos métodos son muy importantes, puesto que son los que permiten modificar y recuperar el valor de los atributos.

```
public   class  NuevoComponente  extends  ComponenteBase  implements
Serializable {
private tipo(int, String,...) nombrePropiedad;
public NuevoComponente(){}
public tipo setNombrePropiedad(tipo nombrePropiedad){
        this.nombrePropiedad=nombrePropiedad;
}
public void getNombrePropiedad(){
        return nombrePropiedad;
}
}
```

Código 8. Métodos get y set.

Desde el menú de propiedades de vista de diseño ya aparecerán, si todo se ha hecho de forma correcta, todos los atributos del nuevo componente.

Por ejemplo, en el siguiente caso, se ha creado un nuevo componente basado en JTextField que incorpora una nueva propiedad booleana bajo el nombre booleanColor.

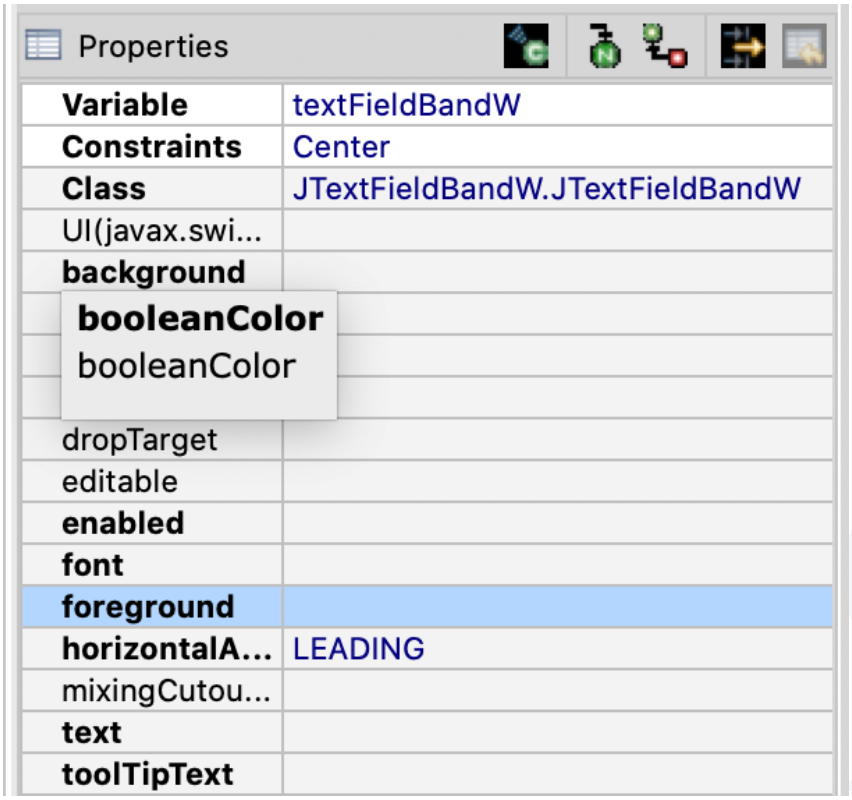

Fig. 5. Nuevo componente basado en un JTextField.

/ 8. Integración de un nuevo componente. Empaquetado

Para agregar un nuevo componente a la paleta de diseño, bien desarrollado por nosotros, o implementado por otros, en primer lugar, es necesario disponer del **Jar** del componente.

En el caso de utilizar el nuevo componente creado en los apartados anteriores, los pasos para generar un *Jar* desde Eclipse son:

1. Crea un paquete **(package)** y coloca la clase o clases implementadas del nuevo componente dentro del paquete.

2. Desde el menú de herramientas, limpiamos y construimos el proyecto. **(Project>Clean y Project>Build project).**

3. A continuación, accedemos a la opción **Export** del proyecto donde se ubica el nuevo componente (pulsando con el botón derecho sobre el nombre del proyecto) y escogemos la opción **Jar File**, situada dentro de la carpeta Java. Finalmente, indicamos un nuevo nombre para el archivo *Jar* y se guarda.

Ahora, para incluir el componente en un nuevo proyecto, en primer lugar, se añade a las librerías el archivo Jar creado o descargado. Esto se realiza a través de la opción **Build Path** (botón derecho sobre el nuevo proyecto) y, a continuación, *Configure* **Build Path**. Finalmente, sobre **Classpath**, añadimos el *Jar* utilizando la función **Add external jars.**

Por último, se incluye el componente en la paleta de la zona de diseño, *Palette* (aunque el proceso de incorporación se realiza desde un proyecto específico), y el nuevo elemento quedará incluido de forma permanente.

Con el botón derecho, pulsamos sobre la categoría donde se va a incluir el nuevo componente y, a continuación, **Add component.**

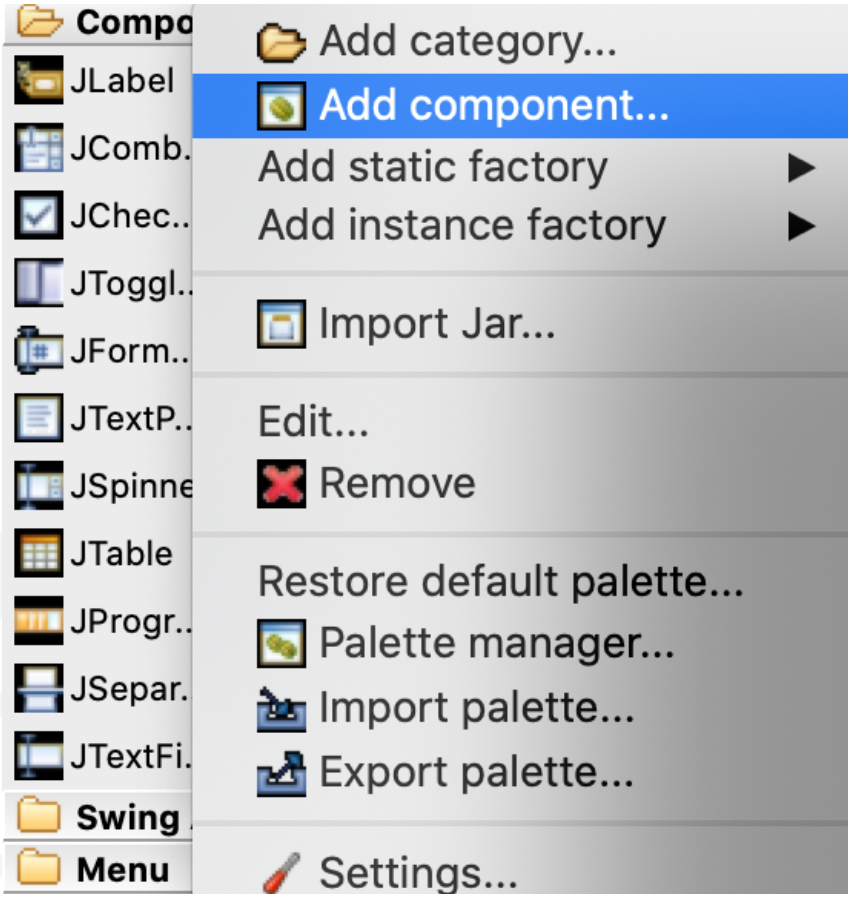

Fig. 6. Añadir un nuevo componente.

Si la inclusión de la librería se ha realizado correctamente al buscar el nombre de la clase, este debe aparecer. Se selecciona y se pulsa OK en las dos ventanas.

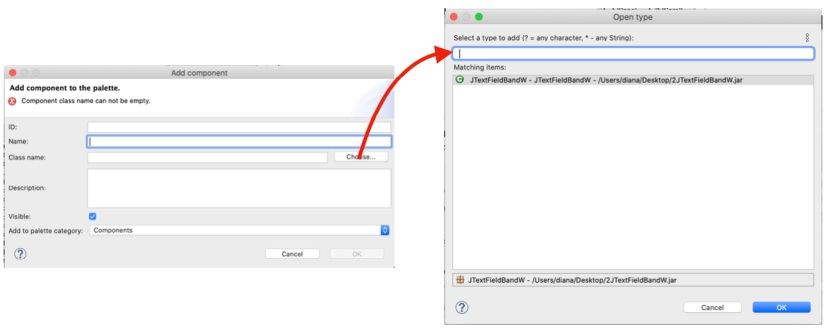

Fig. 7. Búsqueda de la clase del nuevo componente.

Vídeo 1. "Cómo añadir un componente gráfico al modo Design de Eclipse"
https://bit.ly/30VnJGZ

/ 9. Herramientas para el desarrollo de componentes visuales

El desarrollo y creación de elementos visuales para su inclusión en el diseño de una interfaz requiere del uso de herramientas para el diseño gráfico.

Existen multitud de herramientas en la actualidad que pueden utilizarse para la edición de imágenes, desde algunas más profesionales como Adobe Photoshop hasta otras que permiten realizar un amplio conjunto de acciones, suficientes para el desarrollo de gráfico de ciertos componentes. Veamos algunas de ellas:

A) GIMP

En primer lugar, podemos utilizar GIMP. Se trata de un programa de edición de vídeo gratuito. Permite la edición de imágenes digitales en forma de mapa de bits.

B) Paint o Pinta

Microsoft Paint o Pinta son herramientas similares entre sí, que incluyen opciones sencillas que permiten una edición de calidad para aquellos casos en los que no se necesita algo demasiado profesional. Microsoft Paint incluye, en su nueva versión, funcionalidades para el diseño 3D.

C) Photoshop

Adobe Photoshop es una de las herramientas para la edición de imágenes más reconocidas y utilizadas en la actualidad. Una de las funcionalidades más importantes que incorpora esta aplicación es el uso de capas, mediante las cuales es posible aplicar a la imagen multitud de efectos y tratamientos. Algunas de sus características son:

- **Elevada potencia de procesamiento** de imágenes, lo que la hace muy adecuada para el entorno profesional.
- Permite elaborar **diseños desde cero.**
- Incluye **un sistema de capas** que permite crear imágenes muy atractivas.
- Soporta muchos tipos de **formatos.**
- **Sistema de filtros**, efectos, eliminación del ruido, retoque de la imagen...

Fig. 8. Logo GIMP.

Fig. 9. Logo Photoshop.

/ 10. Graphics y figuras

Además de la inclusión de componentes ya predefinidos, a través de la clase **Graphics de Java es posible dibujar** sobre la interfaz manejando los píxeles como si de un lienzo en blanco se tratara. Graphics se encuentra bajo la librería AWT. Dos de los métodos esenciales para dibujar son:

paint (Graphics nombre)	update (Graphics nombre)
Se utiliza para dibujar sobre la interfaz la primera vez que se muestra, cuando se maximiza, o cuando vuelve a estar visible. Todos los componentes utilizan este método para dibujar su "forma". Si se crea un nuevo componente a partir de otro y se quiere modificar su diseño gráfico, se debe sobrescribir este método.	Se utiliza para actualizar los gráficos dibujados sobre la interfaz.

Tabla 2. Métodos de la clase Graphics

Uno de los elementos más importantes de esta clase es el atributo Color, el cual, a través del método **setColor (Color c)**, permite definir el trazo de la figura que se va a diseñar.

Por otra parte, es posible añadir figuras geométricas sobre la interfaz de diseño. Los métodos utilizados son *draw* y *fill*. El primero dibuja el perímetro de la figura indicada, mientras que el segundo, además, lo rellena del color señalado.

drawLine (int x1, int y1, int x2, int y2)	drawRec (int x, int y, int width, int height) fillRec (int x, int y, int width, int height)	drawOval (int x, int y, int width, int height) fillOval (int x, int y, int width, int height)	drawPolygon (int [] x1, int [] y1, int [] x2, int [] y2) fillPolygon (int [] x1, int [] y1, int [] x2, int [] y2)
Dibuja una línea desde la posición inicial marcada por las coordenadas x1 e y1 hasta la final x2 e y2.	Dibuja un rectángulo desde la posición inicial marcada por las coordenadas x e y, y con la altura y ancho indicados. En el caso del segundo método, se rellena del mismo color que el definido para la línea del borde.	Dibuja una elipse desde la posición inicial marcada por las coordenadas x e y, y con la altura y ancho indicados. fillOval crea y rellena la elipse del mismo color que el definido para la línea del borde.	Dibuja un polígono utilizando el array de coordenadas x e y pasado por parámetro. De la misma forma que en casos anteriores, fillPolygon realiza el diseño relleno del mismo color que la línea de perímetro.

Tabla 3. Métodos para crear figuras.

/ 11. Imágenes

Los componentes destinados a la creación de ventanas, paneles u otros contenedores no incluyen la opción de **cargar a través de una URL una imagen de fondo**. Ahora bien, utilizando los métodos propios de la **clase Graphics** y añadiendo ciertas modificaciones sería **posible incorporar** esta funcionalidad, hablamos del método *paint*.

En el siguiente fragmento de código, en primer lugar, se realiza un llamamiento al **método paint** utilizando la palabra reservada super, por lo tanto, se está haciendo un llamamiento al constructor de la superclase.

```
super.paint(g);
```

A continuación, se crea una instancia de la clase *Toolkit*, esto permite asociarle una imagen a través de su URL completa a un objeto de tipo *Image*.

```
Toolkit t = Toolkit.getDefaultToolkit ();
Image imagen = t.getImage ("rutaCompleta");
```

Para mostrar la imagen cargada se utiliza el método *drawImagen* de la clase *Graphics*, el cual recibe, también por parámetro, la posición exacta en la que la imagen se va a dibujar.

```
g.drawImage(imagen, coordenadaX, coordenada Y, ancho, altura, this);
```

Si no se indica el valor correspondiente a las dimensiones de la imagen, se cargará la imagen con el tamaño con la que ha sido guardada. Es aconsejable ajustarlo a la ventana, puesto que de lo contrario la imagen se vería incompleta.

En el siguiente fragmento de código se ha utilizado una imagen cuyas dimensiones exceden el tamaño de la ventana, por lo que se han ajustado al tamaño del frame maximizado.

```
public void paint(Graphics g){
        super.paint(g);
Toolkit t=Toolkit.getDefaultToolkit();
Image i = t.getImage("…/imagen.jpg");
int ancho=(int)(t.getScreenSize().getWidth());
int alt=(int)(t.getScreenSize().getHeight());

g.drawImage(i, 0, 0, ancho, alt, this);
}
```

Código 9. Ejemplo de inclusión de imagen ajustando su tamaño.

Vídeo 2. "Interfaz de edición de márgenes de fotografías con la clase Graphics"
https://bit.ly/2X3cICu

/ 12. Casopráctico 1: "Interfaz metro"

Planteamiento: Se ha recibido un encargo de la Red de Transporte Regional de Metro para desarrollar una interfaz que permita recargar el abono mensual de transporte. Para que la recarga sea rápida e intuitiva, la interfaz debe tener un menú desplegable con los meses del año para poder seleccionar el mes que se quiere recargar en el abono.

Nudo: La mejor manera de controlar la fecha introducida en una interfaz es con un calendario en el que el usuario solo tenga que hacer clic para seleccionar el año, mes o día. De esta forma, además, es sencillo poder recuperar los valores seleccionados para trabajar con ellos. En este caso, como la librería *Java Swing* no trae por defecto este componente, se ha importado la librería *JCalendar* para poder utilizar el componente *JMonthChooser*.

Para agregar un nuevo componente a la paleta de diseño, en primer lugar, es necesario disponer del **Jar** del componente. Los pasos necesarios a seguir son:

1. Incluir el componente en el proyecto, añadiendo las librerías del archivo Jar descargado. Desde la opción **Build Path, Configure Build Path**. Sobre **Classpath** añadimos el *Jar* utilizando la función **Add external jars**.

2. Incluir el componente en la paleta de la zona de diseño. Para ello, con el botón derecho, pulsamos sobre la categoría donde se va a incluir el nuevo componente y, a continuación, **Add component**.

3. Declarar el componente.

```
private JMonthChooser monthChooser;
```

4. Instanciar el componente.

```
private JMonthChooser monthChooser;
```

5. Ubicarlo y agregarlo al panel.

```
monthChooser.setBounds...
panel.add(monthChooser);
```

Desenlace:

Fig. 10. Componente JMonthChooser del caso práctico 1.

/ 13. Caso práctico 2: "Formas básicas de la clase Graphics"

Planteamiento: En este ejercicio se va a practicar con un óvalo verde, un rectángulo amarillo y un triángulo naranja realizados con la clase Graphics de Java. Partimos de un código base de un JFrame con un constructor del siguiente tipo:

```
public class Grafico1 extends JFrame {
 private JPanel contentPane;
 public static void main(String[] args) {
 EventQueue.invokeLater(new Runnable() {
  public void run() {
    try {
      Grafico1 frame = new Grafico1();
      frame.setVisible(true);
    } catch (Exception e) {
        e.printStackTrace();
...
public Grafico1({
  setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
  setBounds(100, 100, 450, 300);
  contentPane = new JPanel();
  contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));
  setContentPane(contentPane);
  contentPane.setLayout(null);
  setBounds(0,0,800,600);
}
```

Código 10. Código base caso práctico 2.

Nudo: Los pasos a seguir para realizar este ejercicio serán:

1. Sobrescribir el método paint.
2. Llamar al método paint de la clase superior.
3. Activar el método del color que queramos, en este caso es verde.
4. Dibujar un óvalo desde la coordenada 'x' en 150 píxeles e 'y' en 70 píxeles. Con un radio en el eje 'x' de 50 píxeles y de 70 píxeles en el eje 'y'.
5. Cambiar el color a amarillo.
6. Dibujar un rectángulo que comience en el eje 'x' en el píxel 350 y en el eje 'y' en el 70. De ancho que tenga 50 píxeles y de alto 70.
7. Cambiar el color a naranja.
8. Dibujar un triángulo utilizando el método fillPolygon indicando las posiciones en el eje 'x' de los tres vértices, las posiciones de los tres vértices en el eje 'y'.

Desenlace:

```
public void paint (Graphics g){
super.paint(g)
g.setColor (Color.green);
g.fillOval (150, 70, 50, 70);
g.setColor (Color.yellow);
g.fillRect (350, 70, 50, 70);
g.setColor (Color.orange);
int [] vx1 = {600, 650, 550};
int [] vy1 = {70, 120, 120};
g.fillPolygon (vx1, vy1, 3);
}
```

Código 11. Código desenlace caso práctico 2.

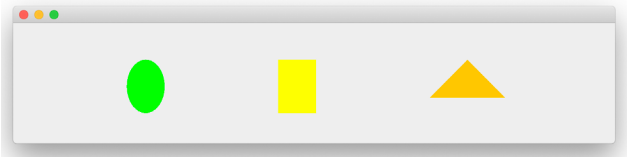

Fig. 11. Interfaz con las principales formas de Graphics del caso práctico 2

/ 14. Resumen y resolución del caso práctico de la unidad

A lo largo de este tema, se han expuesto cuáles son las características principales que debe presentar todo **componente**, esto es clave para comprender qué aspectos son esenciales en la creación de nuevos elementos. A través de la implementación de componentes de ejemplo de manera práctica, se ha expuesto el **flujo común principal de desarrollo.**

De la misma forma, la distribución y posterior integración de aquellos componentes que no se recogen en la paleta básica de entorno de desarrollo, resultan esenciales para la creación de **entornos de desarrollo personalizados** que se ajusten a las especificaciones de cada nuevo proyecto.

El diseño de interfaces no solo requiere de elementos con una funcionalidad específica, como los botones, los menús o las cajas de texto, sino que serán útiles para mejorar la experiencia de uso de otros elementos más visuales, como los que se pueden obtener a través de la librería **Graphics**, la cual nos permite "dibujar" sobre la ventana multitud de formas y otros elementos gráficos.

Resolución del caso práctico inicial

Durante el desarrollo de una interfaz, podemos hacer uso de los componentes visuales de los que disponga nuestro entorno de desarrollo o crear nuestros propios componentes. Para resolver el caso inicial planteado, existen varias opciones:

En primer lugar, sería posible utilizar componentes ya existentes en la paleta de Eclipse para la realización de un calendario a través de la combinación de *JTextField y JLabel*. Los de tipo *JTextField* serán cajas de dos caracteres para los días y los meses, y de cuatro caracteres para los años. Los *JLabel* se pueden utilizar para separar los campos de las fechas con un símbolo como "/" o "_".

Por otro lado, es posible utilizar un componente con la funcionalidad de la fecha ya incorporada, estamos hablando de componentes externos que se pueden importar a nuestro entorno. Este es el caso de *JDateChooser*, que permite hacer clic en el icono de un calendario y seleccionar la fecha deseada.

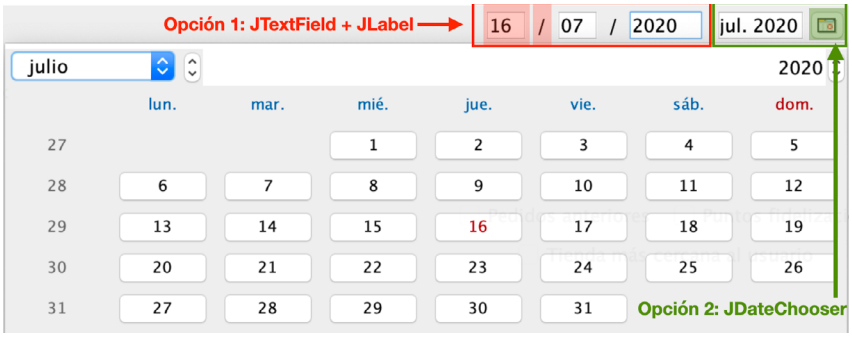

Fig. 12. Opciones para el componente calendario.

/ 15. Bibliografía

Fernández, A., García-Miguel, B. y García-Miguel, D. (2020). Desarrollo de Interfaces (1.a ed.). Madrid, España: Síntesis.