DESARROLLO DE INTERFACES

TÉCNICO EN DESARROLLO DE APLICACIONESMULTIPLATAFORMA

Creación de componentes

Un componente es un elemento **JavaBean**, con características que definen su funcionamiento (introspección, reflexión, customización y comunicación activa entre el usuario y la interfaz a través de los eventos).

Para agregar un componente se genera un archivo .jar que puede ser importado en cualquier entorno de desarrollo, permitiendo así el uso de sus componentes.

Librería Graphics: realiza múltiples diseños gráficos sobre ventanas y paneles.

Es posible crear **nuevos componentes** a partir de uno ya existente: modificando sus características y propiedades, adaptando sus elementos.

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**.

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

Utilizando como base **elementos ya desarrollados** se podrán crear otros nuevos que presenten otras características que el componente tomado como referencia no incorpora. Se toman **como base los ya implementados** y se les incorporan ciertas **mejoras**.

Beneficios de **reutilizar módulos** software:

- **Se reducen los costes** del proyecto (coste = tiempo de desarrollo).
- Las **pruebas** de software se **simplifican**, solo será necesario someter a pruebas a los **nuevos desarrollos**.
- Mejora la **calidad** del software. Cada nueva implementación, se agregan nuevas funcionalidades más específicas.

Reutilización de software, **tres niveles**:

1- a nivel de **clase** (clases y algoritmos).

2- a nivel de **diseño** (patrones de diseño).

3- a nivel de **arquitectura**.

Características de los componentes

Definición: Módulo de código ya **implementado y reutilizable** que puede interactuar con otros componentes software a través de las interfaces de comunicación.

De componentes más generales se podrán **implementar** múltiples **versiones modificadas** (comenzar con componentes software generales...).

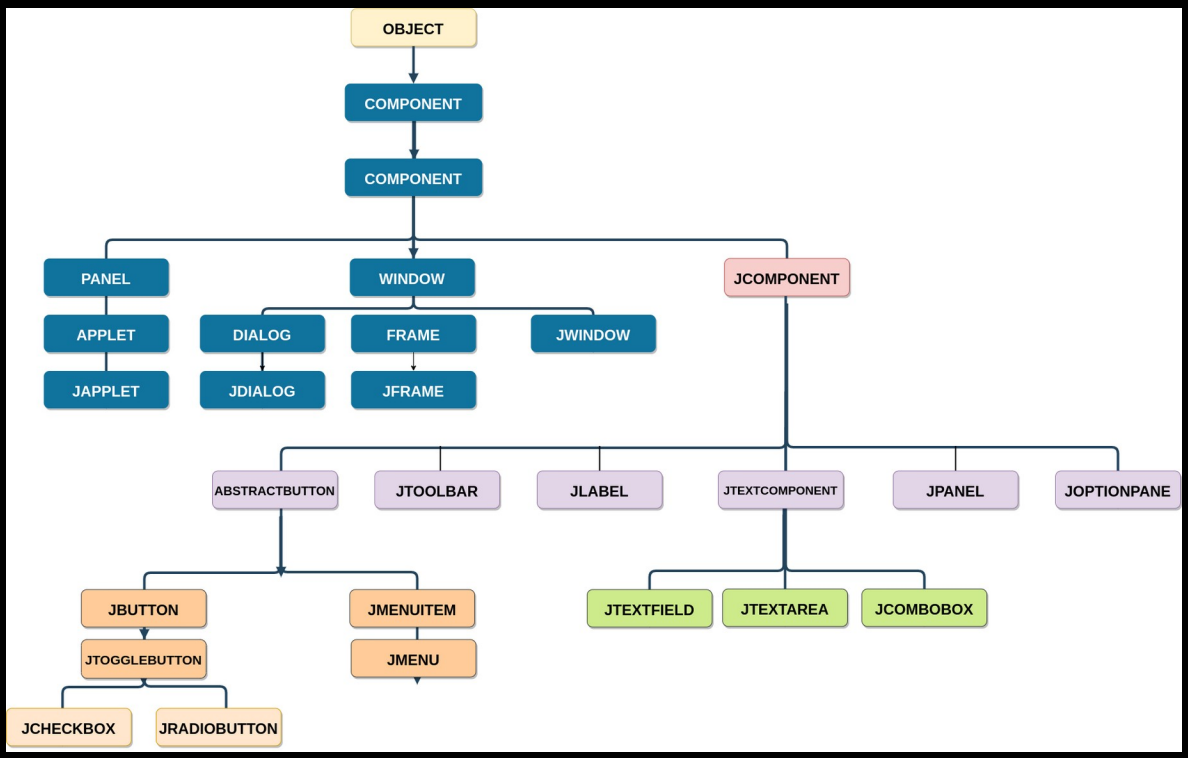
En diseño gráfico con **JSwing**, un **componente** es alguno de los **elementos** que se sitúan en la ventana, directamente sobre el **JFrame** y **JDialog**, o sobre un **JPanel**, y que le aporta funcionalidad a la interfaz.

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** (archivo .jar en java).
- Distribución: **paquete ejecutable**.

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

Componentes visuales (librería **Jswing)** (reutilizables y personalizables):



Propiedades / 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.

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**: propiedad utilizada desde cualquier parte de la **aplicación**.
- **Ámbito privado**: propiedad solo es accesible desde la clase donde se ha creado.
- **Ámbito estático**: propiedad utilizada **sin** la necesidad de crear una **instancia del objeto** al que está referida.

Se distinguen principalmente dos **tipos de propiedades**:

- Las **propiedades simples** (un solo v**alor**):

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

- Las **propiedades indexadas** (**conjunto de valores** en forma de **array)**:

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

Los **atributos** se utilizan para **almacenar los datos internos** y de **uso privado** de una **clase** u objeto.

JavaBean

Son componentes software que permiten su **reutilización en las interfaces de usuario java**. Pueden estar representados gráficamente o no. **Componentes y elementos de la paleta** gráfica (entorno de diseño) son **módulos de tipo JavaBean**.

Es el primer modelo para el desarrollo de componentes de Java e implementa el modelo Propiedad-Evento-Método, sus **características distintivas** son:

- los métodos,
- las propiedades
- y los eventos.

Características comunes:

• **Introspection.** El IDE 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.** Podrá ser almacenado para ser utilizado posteriormente en **cualquier proyecto**. Se logra gracias a la **serialización** del componente (**implements Serializable**).

• **Customization.** **Propiedades** y **comportamiento** modificable en su implementación.

• **Events**. Podrá tener **asociadas** unas **acciones** como respuesta a un estímulo. Es posible la **comunicación** entre **componentes desarrollados** de forma completamente **independiente**.

Diferencia entre un JavaBean y las clases de Java

Un JavaBean es una clase de Java que **agrupa en un objeto a varios objetos** como si fuese una especie de cápsula. Tiene que cumplir las siguientes **reglas**:

- Debe tener un **constructor público sin argumentos**.
- **Ninguna variable** de instancia **pública**.
- Las propiedades deben ser accesibles con los **métodos get y set**.

Los JavaBeans se crearon en un primer momento **para las herramientas gráficas** que se utilizan en el desarrollo de aplicaciones, pero esa tendencia ha cambiado **últimamente** ya que ahora se usan en muchos tipos de programas para **encapsular código** y que sea **reutilizable**.
Un Bean puede tener **clases asociadas**, por ejemplo, una clase BeanInfo proporciona información sobre el Bean, sus propiedades y sus eventos.
Un JavaBean es una clase Java, pero **una clase Java no tiene por qué ser un JavaBean**.

Creación de un nuevo componente JavaBean

Utilizar otros **componentes ya desarrollados** e introducirles **mejoras** y otros **cambios** que se adapten a nuevas casuísticas.

Es necesario crear un **nuevo proyecto** para la creación de un **nuevo componente**. El nuevo componente extenderá del base:

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

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

1. Tiene que implementar **Serializable**:

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

2. Ha de tener un **constructor sin parámetros**:

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

Cada nuevo comportamiento implementa una nueva funcionalidad (**métodos**).

Redefinir el comportamiento: utilizar la palabra reservada **@Override** seguido del mismo **nombre del método** en la **clase de referencia** (“nuevoMetodo” en el ejemplo):

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

Para 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**.

A los nuevos componentes **solo habrá que añadir las propiedades nuevas** porque ya parten con las del componente base. Se **implementan los atributos nuevos**, en la **parte superior** de la clase.

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

Se deberán incorporar los **métodos set y get** a la clase, y así poder establecer y obtener el valor de los atributos.

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

En el **menú de propiedades** de **vista de diseño** ya aparecerán los **atributos** del nuevo componente.

Integración de un nuevo componente.
Empaquetado

Para agregar un nuevo componente **a la paleta de diseño** es necesario disponer del **Jar** del componente base.

Los pasos para generar un Jar desde Eclipse son:

1. Crea un **paquete** 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

    Project>Build project

3. 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 bien, antes de poderlo utilizar en otros proyectos debemos de **modificar el archivo manifest** generado en el paquete del componente de la siguiente manera: descomprimimos el paquete .jar; dentro, en el directorio META-INF encontramos el manifest.mf, que contiene solamente: *Manifest-Version: 1.0*. Debemos dejar una línea en blanco debajo de esta única línea que ya contenía, y a continuación en la siguiente línea escribir:

Name: [y seguido, sin estos corchetes, la ruta completa de nuestra clase bean dentro del componente nuevo]

por ejemplo, si la clase en la que creamos nuestro componente se llama *NuestroComponenteNuevo.class*, sería:

Manifest-Version: 1.0.

Name: beans/NuestroComponenteNuevo.class

Java-Beans: true

Dejando la última línea en blanco, después de *Java-Beans: true.*

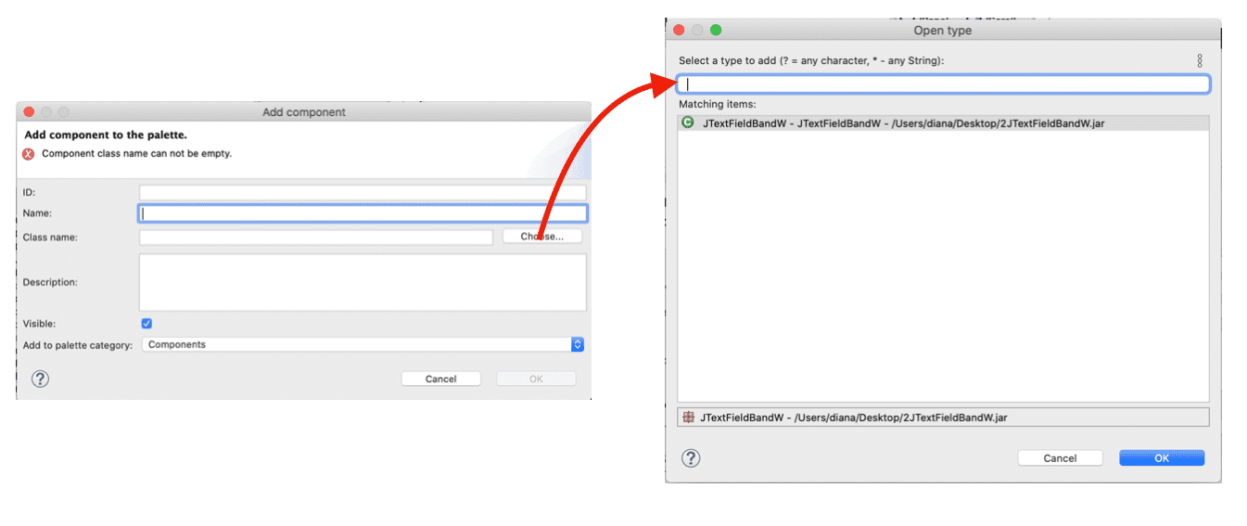
Cambiando este archivo manifiesto con el nuevo archivo ya modificado, ya tenemos el nuevo componente bean .jar listom para usar en nuestros proyectos

Para incluir el componente en un nuevo proyecto, 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**.

Y se **incluye el componente en la paleta** de la zona de diseño, Palette 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**.

Ahora, al **buscar el nombre de la clase**, este debe **aparecer**. Se selecciona y se pulsa OK en las dos ventanas.

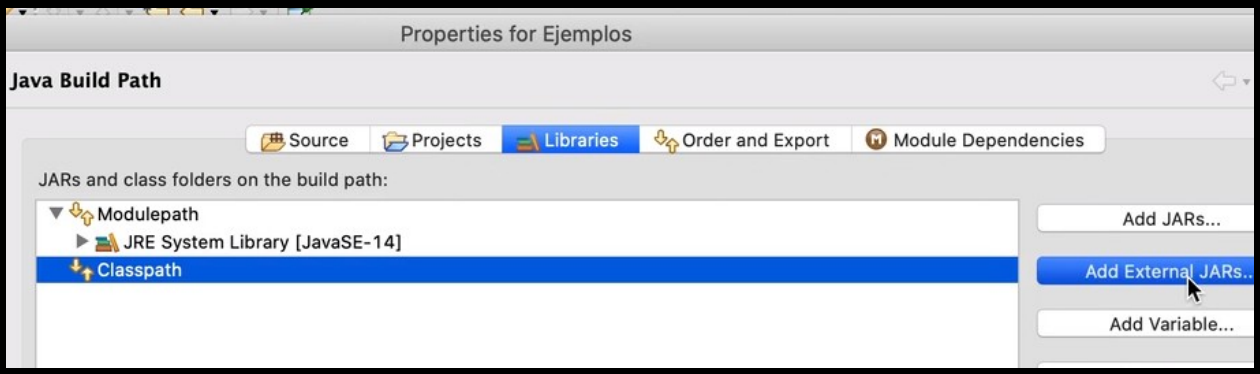


Ejemplo:

Añadir un calendario con Jcalendar, de https://toedter.com/jcalendar/:

Leemos su API... y descargamos su paquete. En el directorio lib (librerias) buscamos los archivos .jar que son los que nos interesan, y todos esos, sean los que sean, son los que importaremos a nuestro proyecto.

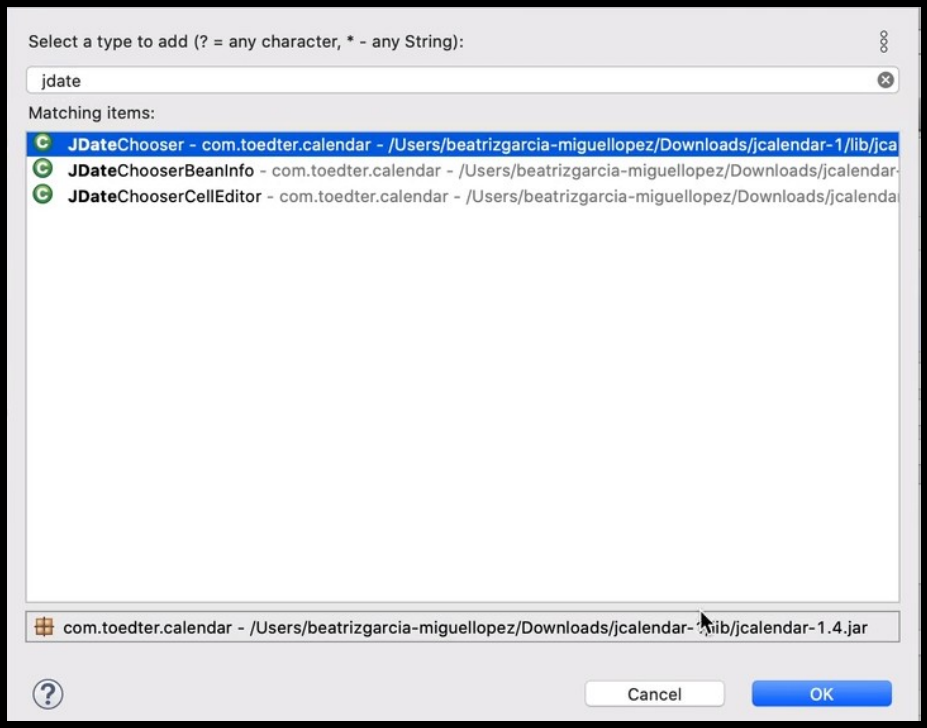
Y en nuestro proyecto, botón derecho de ratón y en el menú contextual: ***build path***, después ***configure build path***, y en la ventana que se abre, ***Add External JARs***:



Seleccionamos los archivos .jar que tenemos guardados.

Luego ya podremos importar el paquete JDateChooser, que se incluye en las librerías que acabamos de agregar al proyecto.

Para añadirlo como componente a la paleta, haciendo click derecho sobre componentes, añadir componente, buscamos la clase: JdateChooser. Y ya nos aparecerá en nuestra paleta.



Herramientas de desarrollo de componentes visuales

GIMP

Programa de edición de imágenes gratuito. Edición de representaciones digitales en forma de mapa de bits, dispone de tijeras inteligentes, filtros, la posibilidad del uso de **capas**, entre otras **muchas características**.

Microsoft Paint o Pinta

Permiten una edición de calidad para aquellos casos en los que **no** se necesita algo **demasiado profesional**. Funcionalidades para el diseño 3D.

Photoshop

Algunas características:

- Elevada **potencia de procesamiento** de imágenes.
- Elaborar diseños desde cero.
- Sistema de **capas** (multitud de efectos y tratamientos).
- Muchos tipos de **formatos**.
- Filtros, efectos, eliminación del ruido, retoque de la imagen…

Graphics y figuras

Permite 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): 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.

update(Graphics nombre): Actualiza los gráficos dibujados sobre la interfaz.

El **atributo Color** a través del método **setColor** (Color c), permite **definir el trazo** de la figura que se va a diseñar.

Para añadir figuras geométricas sobre la interfaz de diseño, tenemos los métodos:

drawLine (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.

drawRec (int x, int y, int width, int height): Dibuja un rectángulo desde la posición inicial marcada por las coordenadas x e y, y con la altura y ancho indicados.

fillRec (int x, int y, int width, int height): Crea y rellena el rectángulo del mismo color que el definido para la línea del borde.

drawOval (int x, int y, int width, int height): Dibuja una **elipse** desde la posición inicial marcada por las coordenadas x e y, y con la **altura y ancho** indicados.

fillOval (int x, int y, int width, int height): Crea y rellena la elipse del mismo color que el definido para la línea del borde.

drawPolygon (int [] x1, int [] y1, int nPoints): Dibuja un polígono utilizando el array de coordenadas x e y pasado por parámetro.

fillPolygon (int [] x1, int [] y1, int nPoints): Dibuja un polígono relleno del mismo color que la línea de perímetro.

Ejemplo de dibujo de formas básicas con Graphics:

Este es el código base de un JFrame con su constructor:

```
import java.awt.EventQueue;
public class DibujoFormasGrafics extends JFrame {

	private static final long serialVersionUID = 1L;
	private JPanel contentPane;

	/**
	 * Launch the application.
	 */
	public static void main(String[] args) {
		EventQueue.invokeLater(new Runnable() {
			public void run() {
				try {
					DibujoFormasGrafics frame = new
							DibujoFormasGrafics();
					frame.setVisible(true);
				} catch (Exception e) {
					e.printStackTrace();
				}
			}
		});
	}
	/**
	 * Create the frame.
	 */
	public DibujoFormasGrafics() {
		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);
	}
}
```

Y sobreescribimos el **método paint**:

```
	@Override
		public void paint(Graphics graphics) {
			// Llamar al método paint de la clase superior.
			super.paint(graphics);
			// Activar el método del color que queramos, en este caso es verde
			graphics.setColor(Color.green);
			/**
			  Dibujar un óvalo desde la coordenada 'x' en 150 píxeles
			  e 'y'
			   en 70 píxeles.
			  Con un radio en el eje 'x' de 50
			 */
			graphics.fillOval(150, 70, 50, 70);
			graphics.setColor(Color.yellow);
			/**
			   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.
			 */
			graphics.fillRect(350, 70, 50, 70);
			graphics.setColor(Color.orange);
			/**
			   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'.
			 */
			int[] vx1 = { 600, 650, 550 };
			int[] vy1 = { 70, 120, 120 };
			graphics.fillPolygon(vx1, vy1, 3);
		}
```

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 **drawImage** 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);
}
```