

---

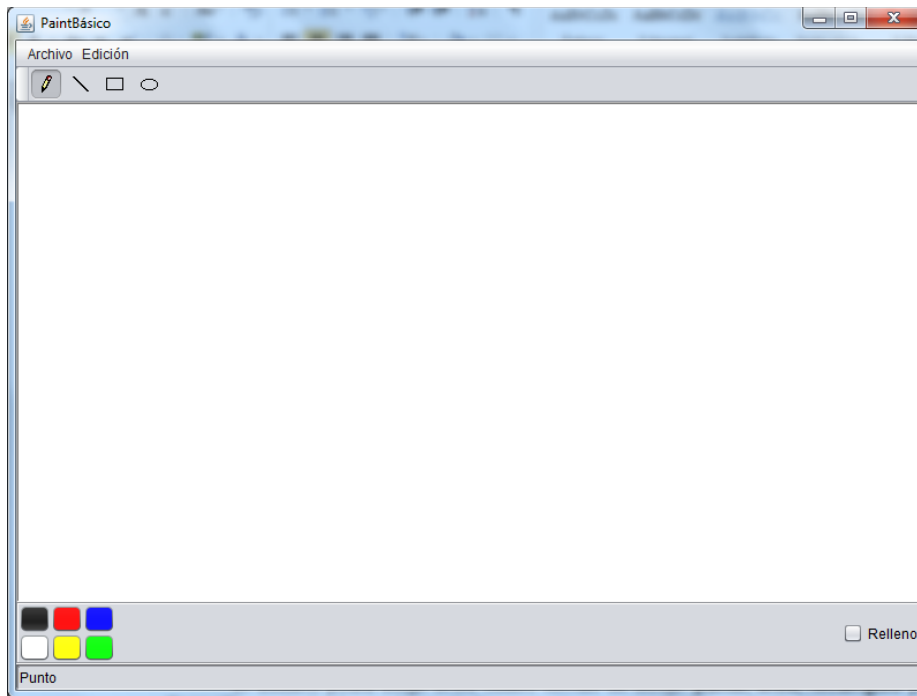
# PRÁCTICA 4

## Ejercicio “Paint Básico”

---

### ■ Descripción del ejercicio

El objetivo de esta práctica es realizar una aplicación sencilla para dibujar formas básicas, pudiendo elegir el color de la misma y si está o no rellena. Para ello, se hará uso de los diferentes elementos Swing vistos en las prácticas anteriores, así como una gestión de gráficos basada en la clase *Graphics*. El aspecto visual de la aplicación será el siguiente:



*Figura 1: Aspecto de la aplicación*

El usuario podrá elegir entre cuatro formas de dibujo: **punto, línea, rectángulo y elipse**. Para ello, se incluirá una barra de herramientas que contenga un botón por cada una de las formas, siendo dichos botones del tipo “dos posiciones” y estando siempre seleccionado aquel correspondiente a la forma activa. La barra de estado también deberá indicar la forma activa.

Además, el usuario podrá elegir entre un **conjunto de colores predeterminados** (mostrados en la parte inferior) y si la forma está o no rellena. En cada momento, el lienzo central mostrará **sólo la última forma dibujada**, no siendo necesario volcar su contenido en una imagen o en un vector de formas.

En el menú se incluirán dos opciones: “Archivo” y “Edición”. La primera tendrá a su vez tres opciones: “Nuevo”, “Abrir” y “Guardar”. La opción “Nuevo” deberá borrar la forma que esté dibujada, mientras que “Abrir” y “Guardar” deberán lanzar el diálogo correspondiente (si bien no se abrirán ni guardarán archivos). El menú edición deberá incluir la opción “Ver barra de estado” que active/desactive la barra de estado.

## ■ Algunas recomendaciones

1. Para el panel de dibujo central, se recomienda crear una clase propia “Lienzo” que herede de *JPanel*. Dicha clase incluiría, entre otros aspectos:
  - Información sobre la forma activa y los atributos (color y relleno) con los que pintar (se recomienda definir métodos set/get para modificar estas propiedades)
  - Método *paint* sobrecargado, donde se incluiría el código para pintar las distintas formas con sus atributos
  - Gestión de eventos de ratón vinculados al proceso de dibujo

Véase apéndice para más detalles.

2. Para lanzar los diálogos de Abrir y Guardar, usar la clase *JFileChooser*. El siguiente código lanzaría el diálogo Abrir (el de cerrar sería igual, pero usando el método *showSaveDialog*).

```
JFileChooser dlg = new JFileChooser();
int resp = dlg.showOpenDialog(this);
if( resp == JFileChooser.APPROVE_OPTION) {
    File f = dlg.getSelectedFile();
    //Código
}
```

## ■ Entrega del ejercicio

La entrega se hará en dos fases:

- Una versión preliminar de la práctica al final de la clase (recogerá lo que se ha hecho durante la sesión de clase, no ha de estar terminada)
- La versión final (ya completa) en la siguiente clase de prácticas

La entrega se hará a través de la web de decsai.ugr.es (se habilitarán dos entregas, una por versión). En ambos casos, habrá que entregar un fichero comprimido (zip o rar) que incluya el proyecto (carpeta Neatbeans) y el ejecutable (.jar)

## ■ Apéndice: Incluir la clase panel propia “Lienzo”

A continuación mostraremos cómo (1) crear una clase propia “*Lienzo*” que herede de *JPanel*, (2) cómo incorporar un objeto de dicha clase a nuestra ventana principal, (3) cómo dibujar en el lienzo recién creado y (4) cómo definir atributos que después puedan ser modificados desde mi ventana principal.

Supongamos el ejemplo de la Figura 2, donde tenemos una ventana principal (*JFrame*) con cuatro botones situados en la periferia de un *BorderLayout*. Supongamos que queremos añadir en el centro de la ventana un panel de una clase propia “*Lienzo*”.

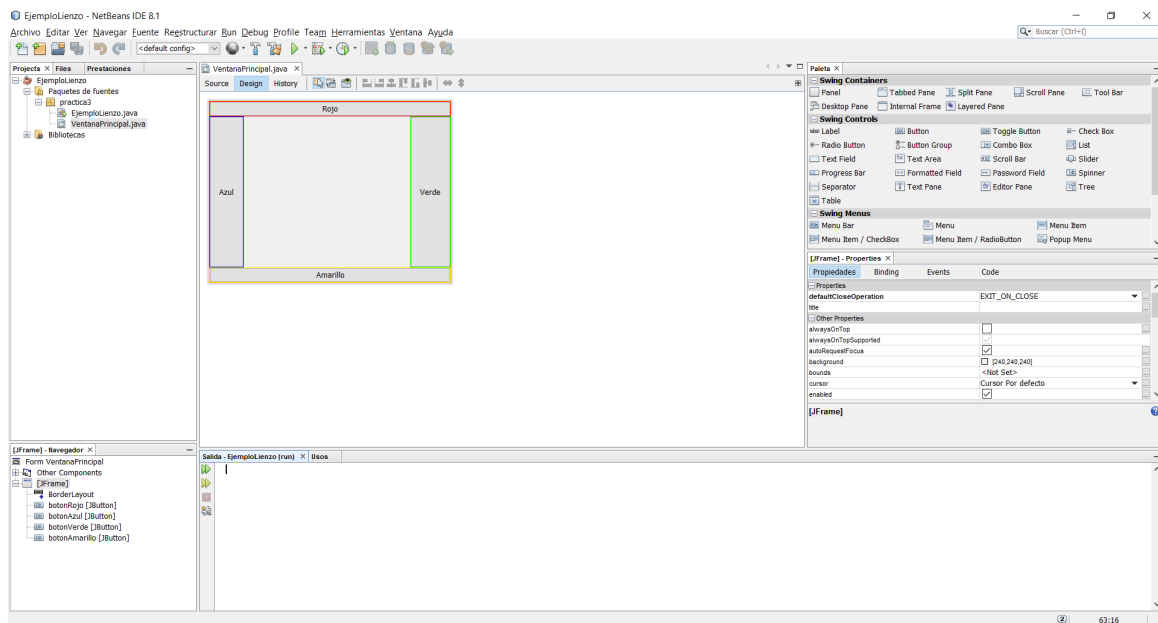



Figura 2: Ventana principal con cuatro botones

### ● Crear la clase Lienzo

Para incorporar un panel propio, es necesario crear una nueva clase que herede de *JPanel*. Para ello, lo mejor es hacerlo usando las plantillas que ofrece NetBeans: esto nos permitirá usar todas las herramientas visuales (componentes, contenedores, eventos, etc.) para diseñar y gestionar nuestro panel<sup>1</sup>.

Por tanto, pulsaremos el botón  (“New File”), seleccionaremos la categoría “Swing GUI Forms” y, de la lista de tipos, elegiremos la opción “*JPanel Form*” (véase Figura 3). A continuación, le indicaremos el nombre de la clase (en nuestro caso: “*Lienzo*”) y, tras aceptar, nos creará un nuevo fichero *Lienzo.java* en nuestro proyecto. Si observamos, dicho fichero tiene una estructura similar al de la ventana principal (un método *initComponents()* y un constructor que llama a dicho método); además, tiene activa la sección “Diseño”, por lo que es posible trabajar con las herramientas visuales e incorporar componentes/contenedores, cambiar manejadores de disposición, gestionar eventos, etc.

<sup>1</sup> Si optásemos por crear la clase sin usar la plantilla que nos ofrece NetBeans, esto es, definiendo nosotros directamente la clase con su correspondiente herencia (`class Lienzo extends JPanel{...}`), no nos aparecería en el editor la sección “Diseño” y, por lo tanto, tendríamos que gestionar todo nosotros. En particular, tendríamos que programar de cero la gestión de eventos (es decir, tendríamos que definir la clase del manejador, sobrecargar los métodos correspondientes, crear el objeto manejador, enlazar generadores con manejadores, etc.). Además, para luego incorporar un objeto de la nueva clase en otro contenedor (p.e., en un *JFrame*), tampoco podríamos usar las herramientas visuales (es decir, no podríamos “arrastrar y soltar” para añadir nuestro panel en cualquier otro contenedor); en su lugar, tendríamos que programarlo nosotros (crear el objeto y añadirlo).

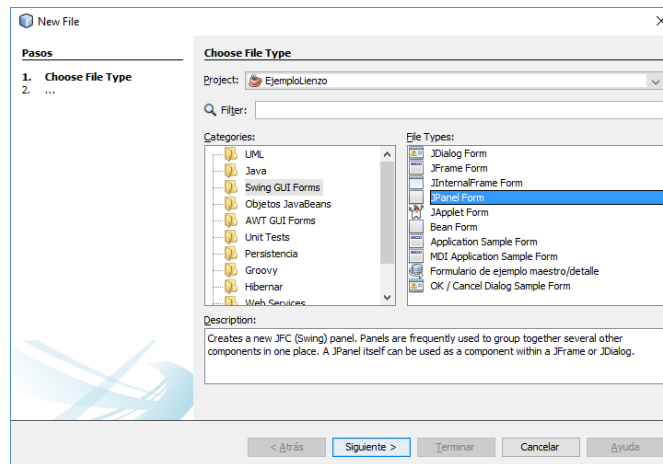


Figura 3: Diálogo para la selección de la plantilla

Una vez creada la clase, para incorporar un nuevo objeto *Lienzo* en el centro de nuestra ventana, basta con que seleccionemos el fichero (árbol de la parte izquierda) y arrastremos y soltemos en el centro de nuestra ventana principal:

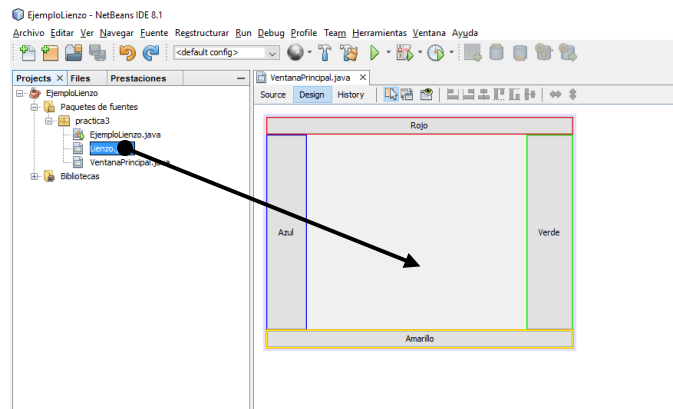


Figura 4: Arrastrar y soltar para añadir un "lienzo" en nuestra ventana

## • Dibujando en Lienzo

Una vez creada la clase *Lienzo*, podemos incorporarle nuevas variables y métodos. En particular, podríamos replicar lo realizado en la práctica 3 para dibujar puntos y líneas sobre un lienzo. El proceso sería exactamente igual, solo que en este caso todo lo haríamos en la clase *Lienzo*: (1) sobrecargaríamos el método `paint(Graphics)`, incluyendo en él el código para dibujar, (2) manejaríamos los eventos de ratón asociados al clic, `pressed`, `released` y `dragged`, y (3) en ambos casos, se usarían variables privadas que almacenarían información relativa a las coordenadas (puntos) donde se ha lanzado cada evento.

Por ejemplo, si sobrecargamos el método `paint(Graphics)` e incluimos el siguiente código:

```
public void paint(Graphics g){
    super.paint(g);
    g.fillOval(50,50,20,20);
}
```

al ejecutar aparecería nuestro lienzo en el centro de la ventana con un punto dibujado en la coordenada (50,50). Si, siguiendo lo visto en la práctica 3, combinamos lo anterior con la gestión de eventos:

```
Point p = new Point(0,0);
.
.
.
public void paint(Graphics g){
    super.paint(g);
    g.fillOval(p.x-10,p.y-10,20,20);
}
.
.
.
private void formMouseClicked(java.awt.event.MouseEvent evt) {
    p = evt.getPoint();
    this.repaint();
}
```

tendremos que, al hacer clic, se irá pintando un punto.

## ● Cambiando atributos: el color

En el caso anterior, el punto siempre se dibuja de color negro. Si quisiéramos cambiar el color, usaríamos el método `setColor(Color)` de la clase `Graphics`:

```
public void paint(Graphics g){
    super.paint(g);
    g.setColor(Color.red);
    g.fillOval(p.x-10,p.y-10,20,20);
}
```

¿Y si quisiéramos que ese color pudiese variar? Tendríamos que considerarlo una propiedad del `Lienzo` y, por tanto, incluir un dato miembro<sup>2</sup> asociado con sus correspondientes métodos set/get:

```
Color color = Color.black;
.
.
.
public void paint(Graphics g){
    super.paint(g);
    g.setColor(color);
    g.fillOval(p.x-10,p.y-10,20,20);
}

public void setColor(Color color){
    this.color = color;
}

public Color getColor(){
    return color;
}
```

---

<sup>2</sup> Esta misma idea se aplicaría para el resto de propiedades del Lienzo. En el ejercicio que nos ocupa, las propiedades son tres: (1) la herramienta de dibujo –punto, línea, rectángulo o elipse–, (2) el color y (3) si se rellena o no la figura. Para el primer caso, lo adecuado sería definir el dato miembro de tipo enumerado; en el segundo, de tipo `Color`; en el tercero, de tipo booleano (y, en todos los casos, sus correspondientes métodos set y get).

Esto permitiría que cualquier otro objeto pudiese mandarle un mensaje al lienzo y cambiar el color con el que se pinta. Por ejemplo, supongamos que en el ejemplo de la Figura 2 quisiéramos que se cambiara el color al pulsar los botones situados en la periferia. En este caso, manejaríamos el evento “*actionPerformed*” asociado a cada botón de la clase “VentanaPrincipal” (es decir, el código estará en el fichero *VentanaPrincipal.java*, no en el *Lienzo.java*) y haríamos que desde el correspondiente método se cambiara el color activo del lienzo:

```
private void botonRojoActionPerformed(java.awt.event.ActionEvent evt) {  
    this.lienzo.setColor(Color.red);  
}  
  
private void botonAzulActionPerformed(java.awt.event.ActionEvent evt) {  
    this.lienzo.setColor(Color.blue);  
}  
  
private void botonVerdeActionPerformed(java.awt.event.ActionEvent evt) {  
    this.lienzo.setColor(Color.green);  
}  
  
private void bAmarilloActionPerformed(java.awt.event.ActionEvent evt) {  
    this.lienzo.setColor(Color.yellow);  
}
```

donde “*lienzo*” es la variable creada por NetBeans al incorporar un nuevo objeto *Lienzo* en el centro de nuestra ventana.

Nótese que, de esta forma, es la ventana principal la que “se comunica” con el lienzo para cambiarle propiedades (color, herramienta, relleno, etc.), y no al revés (es muy importante diseñar la clase *Lienzo* de forma que sea **totalmente independiente** del resto de clases; esto permitirá que podamos usar esa clase en otros programas).

## ● Y para pintar distintas formas...

Para pintar distintas formas, en *paint(Graphics)* habrá que usar el método adecuado de la clase *Graphics*:

- Para el punto: *fillOval*
- Para la línea: *drawLine*
- Para el rectángulo: *drawRect* y *fillRect*
- Para la elipse: *drawOval* y *fillOval*

Por lo tanto, tendremos que usar condicionales (if/else o switch) para que, en función de la herramienta y del tipo de relleno, se llame a uno u otro método.