PRÁCTICA 8

Imagen IO

Descripción de la práctica

El objetivo de esta práctica es realizar una aplicación multiventana para mostrar imágenes y poder pintar sobre ellas. Deberá incluir las siguientes funcionalidades:

- Entorno **multiventana** donde se podrán **abrir** tantas imágenes como se desee, mostrándose cada una de ellas en ventanas internas independientes (una imagen por ventana).
- En una imagen dada, se podrán **dibujar** usando las formas y atributos de la práctica 7
- Las imágenes se podrán **guardar** (incluyendo las formas dibujadas)
- Como reto final, se plantea la **edición** de las figuras ya dibujadas

La Figura 1 muestra el aspecto final de la aplicación. El menú "Archivo" incluirá tres opciones: "Nuevo", "Abrir" y "Guardar", que tendrán también los correspondientes botones en la barra de herramientas. La opción "Nuevo" deberá crear una nueva ventana interna con una imagen en blanco de un tamaño predeterminado (p.e., 500x500); la opción "Abrir" deberán lanzar el correspondiente diálogo y crear una nueva ventana interna que muestre la imagen seleccionada; la opción "Guardar" lanzará el correspondiente diálogo y guardará la imagen de la ventana interna seleccionada.

Para desarrollar este ejercicio, usar como punto de partida el proyecto "*PaitnBásico2D*" de la práctica 7. De esta forma, el entorno incorporará de inicio las barras de formas y atributos, así como las funcionalidades de dibujo[1].

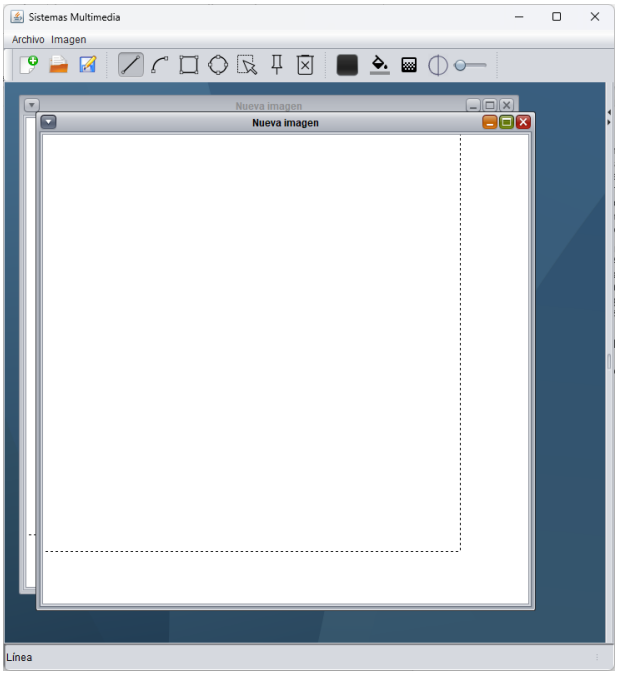


Figura 1: Aspecto de la aplicación

[1] Para mantener una copia del ejercicio "PaintBasico2D" original, en lugar de trabajar directamente en el mismo proyecto, se aconseja copiarlo (botón derecho sobre el proyecto, opción "copiar" en el menú contextual) y ponerle un nuevo nombre tanto al proyecto (p.e. "*SM.PracticasImagen*") como a los paquetes que incluye. Este proyecto lo iremos ampliando en sucesivas prácticas.

Entorno multiventana

Para realizar un entorno multiventana[2], aplicaremos lo siguiente:

- Crear una clase propia `VentanaInterna` que herede de `JInternalFrame` (para ello, usar NetBeans). Activar las propiedades "*closable*", "*iconifiable*", "*maximizable*" y "*resizable*".

  Añadir al centro de esta ventana interna un `Lienzo2D` (el definido para la práctica 7). Esto implicará que podremos dibujar en las ventanas internas. Incluir en `VentanaInterna` un método que devuelva su lienzo:

```
public Lienzo2D getLienzo2D() {
  return lienzo;
}
```

- Anadir un escritorio en el centro de la ventana principal (donde se tuviese el lienzo en la práctica 7, que ahora habrá que quitar). Para ello, incorporar un `JDesktopPane` usando el NetBeans (área "contenedores swing").

- Al quitar el lienzo del centro de la ventana principal, las referencias que hubiese en la práctica 7 a dicho lienzo darán error. En este caso, habrá que cambiar esa referencia al lienzo por una llamada al siguiente método (definido en la ventana principal):

```
private Lienzo2D getSelectedLienzo() {
   VentanaInterna vi;
   vi = (VentanaInterna)escritorio.getSelectedFrame();
   return vi!=null ? vi.getLienzo2D() : null;
}
```

  que devuelve el lienzo de la ventana seleccionada (o `null` si no la hubiese). Nótese que en la llamada al anterior método puede devolver `null`, por lo que habrá que hacer la comprobación cuando vaya a usarse su salida[3].

- Para incorporar una ventana interna al escritorio, hay que (i) crear el objeto, (ii) añadirlo al escritorio y (iii) mandar el mensaje para visualizarla. Así, por ejemplo, en el manejador del evento asociado a la opción "Nuevo", tendríamos:

```
private void menuNuevoActionPerformed(ActionEvent evt) {
  VentanaInterna vi = new VentanaInterna();
  escritorio.add(vi);
  vi.setVisible(true);
}
```

- Si en un momento dado queremos acceder a la ventana que esté activa en el escritorio (p.e., desde un método gestor de eventos), el siguiente código devuelve la ventana activa (`null` si no hay ninguna):

```
VentanaInterna vi;
vi = (VentanaInterna)escritorio.getSelectedFrame();
```

  A través de `vi` podemos acceder, por ejemplo, al lienzo de la ventana activa y a sus métodos y variables.

---

[2] Si tienes dudas, puedes consultar el vídeo tutorial que hay en PRADO.

[3] Donde antes teníamos una referencia a lienzo, por ejemplo `lienzo.setColor(…)`, ahora tendremos:

```
Lienzo2D lienzo = getSelectedLienzo();
if(lienzo!=null){
  lienzo.setColor(…);
}
```

Introduciendo nueva funcionalidad en la clase *Lienzo2D*

Puesto que uno de los requisitos de esta práctica es mostrar una imagen en la ventana interna, es necesario introducir la llamada al método "`drawImage`" en el cuerpo de un método `paint`. La cuestión ahora es en qué clase incorporamos este nuevo código, ¿en una clase nueva específica para mostrar imágenes o, por el contrario, lo hacemos en una ya existente a la que ampliemos su funcionalidad? Ambas opciones tienen sus ventajas e inconvenientes, si bien en esta práctica vamos a optar por la más sencilla: extender[4] la funcionalidad de la clase `Lienzo2D`.

Si recordamos, en la práctica 7 definimos la clase `Lienzo2D` (en particular, es donde sobrecargábamos el método `paint`) [5]. Dado que uno de los requisitos de esta práctica 8 es que se pueda dibujar además de mostrar la imagen, una posible solución sería incorporar a ese lienzo una nueva propiedad: una imagen de fondo. Concretamente, se propone ampliar la clase `Lienzo2D` ya existente de la siguiente forma:

- Añadir una variable de tipo `BufferedImage` que almacenará la imagen de fondo vinculada al lienzo. Se recomienda declarar la variable como privada y definir los métodos `setImage` y `getImage` para modificar el valor de esta variable:

```
public void setImage(BufferedImage img){
  this.img = img;
}

public BufferedImage getImage(){
  return img;
}
```

- El método `paint` deberá visualizar la imagen y la lista de figuras. Para ello, incorporamos la llamada al método `drawimage` antes de pintar el vector:

```
public void paint(Graphics g){
  super.paint(g);
  Graphics2D g2d = (Graphics2D)g;

  if(img!=null) g2d.drawImage(img,0,0,this);

  for(MiShape s: listaFiguras) {
    s.draw(g2d);
  }
}
```

- Para que aparezcan barras de desplazamiento en la ventana interna en caso de que la imagen sea mayor que la zona visible, añadir un `JScrollPane` usando el NetBeans (área "contenedores swing") en el centro de la ventana interna y, dentro del él, añadir el `Lienzo2D`. Además, en el `setImage` habrá que añadir el siguiente código para que el tamaño predeterminado del lienzo sea igual al tamaño de la imagen:

---

[4] Una solución basada en clases desacopladas y con una jerarquía asociada, sería una opción más versátil (por ejemplo, un panel específico para imágenes). No obstante, requeriría conocer con algo más de detalle cómo funciona el método *paint* y a qué otras funcionalidades convoca. Por este motivo, en esta práctica se opta por la solución más sencilla: modificar una clase ya existente. Nótese que esto es posible, en este caso, porque se trata de una clase propia (*Lienzo2D*); si esta clase viniese dada por un tercero (p.e., en una biblioteca), no sería posible modificarla y, por tanto, sería necesario extenderla (herencia) para añadirle las nuevas funcionalidades.

[5] Recordemos que la clase `Lienzo2D` gestiona todo lo relativo al dibujo: vector de formas, atributos, método `paint`, eventos de ratón vinculados al proceso de dibujo, etc. Si se implementó correctamente, dicha clase ha de ser "autónoma" e independiente del resto de clase de la práctica 7. Es decir, podríamos incluir un `Lienzo2D` en cualquier entrono visual de cualquier aplicación y pintar (desde la aplicación, se podrían activar formas y atributos mediante los métodos *set* que ofrece la API de la clase `Lienzo2D`)

```
public void setImage(BufferedImage img){
  this.img = img;
  if(img!=null) {
    setPreferredSize(new Dimension(img.getWidth(),img.getHeight()));
  }
}
```

Nueva imagen

En el gestor de eventos asociado a la opción “nuevo”, además de lanzar una nueva ventana interna, hay que crear una nueva imagen e incorporarla al lienzo:

```
private void menuNuevoActionPerformed(ActionEvent evt) {
  VentanaInterna vi = new VentanaInterna();
  escritorio.add(vi);
  vi.setVisible(true);
  BufferedImage img;
  img = new BufferedImage(300,300,BufferedImage.TYPE_INT_RGB);
  vi.getLienzo2D().setImage(img);
}
```

Con el código anterior, la imagen se verá negra al estar inicializados todos sus pixeles a [0,0,0]. Incluir las líneas necesarias para que la imagen se vea con color de fondo blanco[6].

Abrir imágenes

En el gestor de eventos asociado a la opción de abrir, además del código visto en clase para leer una imagen, hay que incorporar el código necesario para crear la ventana interna y asignarle la imagen recién leída al lienzo:

```
private void menuAbrirActionPerformed(ActionEvent evt) {
  JFileChooser dlg = new JFileChooser();
  int resp = dlg.showOpenDialog(this);
  if( resp == JFileChooser.APPROVE_OPTION) {
    try{
       File f = dlg.getSelectedFile();
       BufferedImage img = ImageIO.read(f);
       VentanaInterna vi = new VentanaInterna();
       vi.getLienzo2D().setImage(img);
       this.escritorio.add(vi);
       vi.setTitle(f.getName());
       vi.setVisible(true);
    }catch(Exception ex){
      System.err.println("Error al leer la imagen");
    }
  }
}
```

Guardar imágenes

En el caso de guardar, habrá que acceder a la imagen de la ventana seleccionada y almacenarla siguiendo el código visto en clase:

[6] No existe un método específico en la clase `BufferedImage` para este propósito. Para poder rellenar la imagen de un color, habrá que acceder a su objeto `Graphics2D` y pintar un rectángulo relleno del color deseado (blanco), cuyas dimensiones coincidan con las de la imagen.

```
private void menuGuardarActionPerformed(ActionEvent evt) {
  VentanaInterna vi=(VentanaInterna) escritorio.getSelectedFrame();
  if (vi != null) {
    BufferedImage img = vi.getLienzo2D().getImagen();
    if (img != null) {
      JFileChooser dlg = new JFileChooser();
      int resp = dlg.showSaveDialog(this);
      if (resp == JFileChooser.APPROVE_OPTION) {
        try {
          File f = dlg.getSelectedFile();
          ImageIO.write(img, "jpg", f);
          vi.setTitle(f.getName());
        } catch (Exception ex) {
            System.err.println("Error al guardar la imagen");
        }
      }
    }
  }
}
```

Obsérvese que con el código anterior se guardaría la imagen, pero no lo que hemos dibujado sobre ella. Es necesario, por tanto, incorporar el código que permita dibujar la lista de figuras sobre la imagen (a través del `Graphics2D` asociado a la imagen). Para ello, se aconseja definir el siguiente método en la clase `Lienzo2D`:

```
public BufferedImage getPaintedImage(){
  // Código para crear una nueva imagen que
  // contenga la imagen actual más las figuras
}
```

Basándose en lo visto en clase, habría que incorporar en el método anterior el código necesario para (1) crear una nueva imagen y (2) dibujar sobre ella la imagen del lienzo más las formas que éste incluye:

- En primer lugar, crearemos una nueva imagen del mismo tamaño y tipo que la original:

```
BufferedImage imgout = new BufferedImage(img.getWidth(),
                                         img.getHeight(),
                                         img.getType());
```

- En segundo lugar, crearemos el "*graphics*" asociado a la nueva imagen para poder pintar sobre ella:

```
Graphics2D g2dImagen = imgout.createGraphics();
```

- Por último, habría que usar `g2dImagen` para pintar en la nueva imagen tanto (1) la imagen de fondo como (2) las distintas formas del vector; además, las formas tendrían que pintarse usando los atributos activos en el lienzo. En principio, el código asociado sería algo como:

```
if(img!=null) g2dImagen.drawImage(img,0,0,this);
for(MiShape s: listaFiguras) {
    s.draw(g2dImagen);
  }
}
```

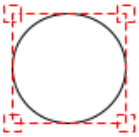
Una vez implementado, en el método `menuGuardarActionPerformed` sustituiríamos la llamada `getImage()` por `getPaintedImage()`.

El reto final…

Por último, se proponen los siguientes retos[7]:

1. **Modo edición: marcar figura seleccionada**. Se incluirá un botón "edición"[8] de dos posiciones dentro de la barra de tareas (agrupado con el resto de las herramientas de figuras), de forma que si está pulsado implicará que estamos en "modo edición" (si se pulsa el botón de una forma de dibujo, automáticamente se saldrá del modo edición).

   Una vez en modo edición, el usuario podrá seleccionar una figura haciendo clic sobre ella. Al seleccionarla, la figura se marcará mediante un rectángulo rojo y discontinuo (su "*bounding box*")[9] con una marca cuadrada en sus esquinas[10].

2. **Editar atributos de la figura seleccionada**. Para la figura seleccionada, se podrán editar sus atributos, esto es, se podrá modificar cualquiera de sus propiedades (color, relleno, etc.) sin más que cambiarla en la barra de herramientas (y los cambios se tendrán que ver reflejados en la forma)[11].

3. **Fijar y eliminar figuras.** En la barra de herramientas incluir dos nuevos botones de dos posiciones: "Fijar" y "Eliminar" (ambos estarán agrupados con el resto de formas de dibujo y con el botón de edición). Su funcionalidad será la siguiente:

   - Cuando esté la opción "fijar" seleccionada, si pasamos por encima de una figura, ésta se "volcará" en la imagen (esto es, se deberá pintar esa figura en la imagen). Nótese que este efecto se producirá simplemente pasando sobre la figura, sin necesidad de seleccionarla o hacer clic sobre ella[12]. Al volcar la figura ésta ya no estará en la lista de figuras del lienzo y, consecuentemente, no aparecerá marcada ni podrá moverse ni editarse (formará parte de la imagen).
   - Cuando esté la opción "eliminar" seleccionada, si pasamos por encima de una figura, ésta se eliminará definitivamente de la lista de figuras. Si una figura ha sido previamente fijada (es decir, ya está "volcada" en la imagen), no podrá borrarse (ya que no será una figura de la lista de figuras).

   ¿Quieres darle un efecto sonoro a tu aplicación? Si te animas, échale un vistazo al pie de página[13].

---

[7] No se incluye en los vídeos. El objetivo es que se analice en qué grado se domina lo aprendido, ¡tú puedes!

[8] Corresponde al botón que se incluyó en la práctica 7 para mover las figuras.

[9] El boundig box de una figura se obtiene mediante la llamada al método *getBounds()*.

[10] Se aconseja incluir una propiedad en las figuras que indique si está o no seleccionada. Añadir además en el método *draw* de la figura el código para pintar la marca en caso de que esté seleccionada. Nótese que será en el lienzo donde se activará/desactivará el estado "seleccionada" de las figuras

[11] Esto implicará que en los métodos *set* del lienzo asociados a atributos habrá que tener en cuenta si estamos en modo edición y, si fuese así, modificar el atributo de la forma seleccionada (si la hubiese).

[12] Es decir, este efecto no se gestiona con el evento *mousePressed*, ni con *mouseClicked* o el *mouseDragged*; ¿sabrías con cuál? Ten en cuenta que deberá de lanzarse simplemente por el hecho de moverse por encima.

[13] Aunque el tema de sonido lo veremos más adelante, puedes seguir los siguientes pasos y hacer que tu aplicación emita un efecto sonoro cada vez que fije o elimine una figura, ¿te apetece probarlo?

a. En el proyecto de tu aplicación, donde tienes la ventana principal, crea un paquete "sonidos" y añade los sonidos que vayas a usar en tu programa (de igual forma que hiciste para los iconos).
b. En tu clase *Lienzo2D* añade dos nuevas propiedades de tipo File:

```
File sonidoFijar, sonidoEliminar;
```

   y añade los correspondientes métodos *setSonidoFijar* y *setSonidoEliminar* para modificarlos.
c. Cuando crees una nueva ventana interna y, con ella, su correspondiente lienzo, actívale los sonidos que quieres que use con el siguiente código:

```
File f = new File(getClass().getResource("/sonidos/fijar.wav").getFile());
vi.getLienzo2D().setSonidoFijar(f);
f = new File(getClass().getResource("/sonidos/eliminar.wav").getFile());
vi.getLienzo2D().setSonidoEliminar(f);
```

   donde "fijar.wav" y "eliminar.wav" son dos sonidos que están en el paquete "sonidos".

Posibles mejoras para trabajar en casa…

Una vez realizada la práctica, se proponen una serie de mejoras para darle mayor funcionalidad y mejorar el interfaz. Si nos centramos en aspectos relativos a lo estudiado en esta práctica, esto es, la creación, lectura y escritura de imágenes, posibles mejoras serían:

1. Usar filtros en los diálogos abrir y guardar para definir tipos de archivos[14].
2. A la hora de guardar, obtener el formato a partir de la extensión del fichero.
3. Lanzar diálogos para informar de los errores (excepciones) producidos a la hora de abrir o guardar ficheros.
4. Permitir al usuario elegir el tamaño de la imagen nueva.

Para mejorar la interfaz de dibujo, además de las ya propuestas en la práctica 7:

5. Definir el área de recorte (clip) del lienzo haciéndola coincidir con el área de la imagen, de forma que no se dibuje fuera de dicha zona[15].
6. Incluir un "marco" (rectángulo) que delimite la imagen.
7. Cuando se cambie de una ventana interna a otra, hacer que los botones de figura y atributos de la ventana principal se activen conforme a la figura y atributos del correspondiente lienzo[16].

---

d. En la clase *Lienzo2D* añade el siguiente método que reproduce el sonido de un fichero "f":

```
private void play(File f) {
  try {
    Clip sound = AudioSystem.getClip();
    sound.open(AudioSystem.getAudioInputStream(f));
    sound.start();
  } catch (Exception ex) {
    System.err.println(ex);
  }
}
```

e. Por último, llama al método *play* anterior donde quieras que se reproduzca un sonido (en nuestro caso, el fichero "f" será sonidoFijar o sonidoEliminar). Donde hagas la llamada dependerá de cómo hayas resuelto el problema de fijar y eliminar figuras

[14] Véase clase *FileFilter* y subclases (por ejemplo, *FileNameExtensionFilter*). Para conocer los formatos reconocibles por *ImageIO*, la clase ofrece métodos como *getReaderFormatNames* o *getWriterFormatNames*

[15] Usar el método `clip` en lugar del `setClip` (el primero combinará la nueva área con la ya existente).

[16] Para abordar esta funcionalidad habrá que gestionar el evento de "ventana interna activa" desde la ventana principal. Véase en PRADO el vídeo tutorial donde se explica cómo hacerlo.