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:

- 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)

El menú "Archivo" incluirá tres opciones: "Nuevo", "Abrir" y "Guardar". La opción "Nuevo" deberá crear una nueva ventana interna con una imagen en blanco de un tamaño predeterminado (p.e., 300x300); 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.

Punto de partida

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].

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[2]. la funcionalidad de la clase `Lienzo2D`.

Si recordamos, en la práctica 7 incorporamos un objeto `Lienzo2D` en el centro de la ventana interna, de forma que focalizamos todo el código para pintar en ese lienzo (en particular, es donde sobrecargábamos el método `paint`) [3]. Dado que uno de los requisitos es que se pueda

---

[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.

[2] Una solución basada en clases desacopladas y con un 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.

[3] 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

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 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 el vector de formas. Para ello, incorporamos la llamada al método `drawimage` antes de pintar el vector[4]:

```
public void paint(Graphics g){
  super.paint(g);
  Graphics2D g2d = (Graphics2D)g;
  if(img!=null) g2d.drawImage(img,0,0,this);
  g2d.setPaint(color);
  g2d.setStroke(stroke);
  for(Shape s:vShape) {
    if(relleno) g2d.fill(s);
    g2d.draw(s);
  }
}
```

- 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:

```
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 (como hacíamos en la práctica 7), 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.getLienzo().setImage(img);
}
```

`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`)

[4] En este ejemplo se está asumiendo que sólo se aplican los atributos de trazo y color; en la práctica serán más los atributos que haya que activar.

Con el código anterior, la imagen se verá negra al estar inicializados todos sus pixeles a [0,0,0]. En el código anterior, incluir las líneas necesarias para que la imagen se vea con color de fondo blanco[5].

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 en el 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.getLienzo().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:

```
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 el vector de formas sobre la imagen (a través del `Graphics2D` asociado a la imagen). Para ello, se aconseja definir el siguiente método en la clase `Lienzo2D`:

[5] 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.

```
public BufferedImage getImage(boolean drawVector){
  if (drawVector) {
     // TODO: Código para crear una nueva imagen
     //       que contenga la imagen actual más
     //       las formas
  }
  else
    return img;
}
```

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);
g2dImagen.setPaint(color);
g2dImagen.setStroke(stroke);
for(Shape s:vShape) {
  if(relleno) g2dImagen.fill(s);
  g2dImagen.draw(s);
}
```

  pero, si observamos, ese código corresponde a cuerpo del método `paint`, así que lo correcto sería llamar a dicho método `paint` pasándole como objeto "*graphics*" el asociado a la imagen. Por tanto, el *if* del método anterior podría implementarse como[6]:

```
if (drawVector) {
    BufferedImage imgout = new BufferedImage( ... );
    this.paint(imgout.createGraphics());
    return(imgout);
}
```

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

---

[6] De cara a futuras prácticas, en las que trabajaremos con imágenes con canal alfa, una mejora en este código sería la siguiente (donde desactiva la propiedad "opaco" del lienzo en caso de que la imagen tenga alfa):

```
if (drawVector) {
    BufferedImage imgout = new BufferedImage( … );
    boolean opacoActual = this.isOpaque();
    if (img.getColorModel().hasAlpha()) {
      this.setOpaque(false);
    }
    this.paint(imgout.createGraphics());
    this.setOpaque(opacoActual);
    return imgout;
}
```

El reto final…

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

- **Efecto ventana**. En la barra de herramientas se incluirá un botón (de dos posiciones) de forma que, cuando esté pulsado, se active el "efecto ventana". Dicho efecto consistirá en activar un área de recorte (por ejemplo, una elipse de tamaño 200x200) que se mueva siguiendo al ratón[8]; con ello, solo se verá aquella parte de la imagen (y figuras) que estén dentro de esa área de recorte[9]. Este efecto se podrá activar o desactivar sin más que pulsar el botón incluido en la barra de herramientas.
- **Volcado de figuras en la imagen**. Incluir un botón en la barra de herramientas que al pulsarlo "vuelque" (pinte) las figuras del vector sobre la imagen (el vector deberá de vaciarse tras la operación). Una vez volcada una figura, ésta ya no podrá moverse (formará parte de la imagen); no obstante, se podrán dibujar nuevas figuras, las cuales sí podrán moverse (mientras que no se repita la operación).

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:

- Usar filtros en los diálogos abrir y guardar para definir tipos de archivos[10].
- A la hora de guardar, obtener el formato a partir de la extensión del fichero.
- Lanzar diálogos para informar de los errores (excepciones) producidos a la hora de abrir o guardar ficheros.
- Permitir al usuario elegir el tamaño de la imagen nueva o, preferiblemente, cambiar el tamaño (redimensionar y/o escalar) de la imagen activa.

Para mejorar la interfaz de dibujo (lienzo):

- 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[11].
- Incluir un "marco" (rectángulo) que delimite la imagen.
- Cambiar el puntero en función de lo que se esté haciendo; por ejemplo, "punto de mira" para pintar en el lienzo, "flecha" para seleccionar, "mover" cuando se desplace la forma, etc.
- A la hora de mover una figura, que el "punto ancla" sea el punto donde se hace el clic al seleccionar (y no la esquina superior, i.e., evitar el "salto" de la práctica 5)

Por último, recordar las mejoras ya propuestas en la práctica 7 para la ventana principal:

- Mejorar las barras de herramientas, simplificándolas con un diseño del tipo:

- Incluir descripciones emergentes ("*tooltips*")
- Cuando se cambie de una ventana interna a otra, hacer que los botones de la ventana principal se activen conforme a la forma y atributos del lienzo.
- Mostrar en la barra de estado las coordenadas del puntero al desplazarse sobre el lienzo.

---

[7] No se incluyen en los vídeos. El objetivo es que se analice en qué grado se domina lo aprendido, ¡tú puedes!
[8] Véase práctica 5.
[9] Para evitar efectos "extraños" con el recorte cuando hay una imagen, se aconseja usar el método "*clip*" en lugar del "*setClip*" para activar el área de recorte.
[10] Véase clase *FileFilter* y subclases (por ejemplo, *FileNameExtensionFilter*). Para conocer los formatos reconocibles por *ImageIO*, la clase ofrece métodos como *getReaderFormatNames* o *getWriterFormatNames*
[11] Usar el método `clip` en lugar del `setClip` (el primero combinará la nueva área con la ya existente).