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].

La clase *LienzoImagen2D*

Para gestionar la imagen situada en la zona central de una ventana interna, se recomienda crear una clase propia `LienzoImagen2D` al estilo de las clases "lienzo" de prácticas anteriores; siguiendo la filosofía iniciada en la práctica 7, se aconseja incorporar la nueva clase al paquete *sm.xxx.iu* de nuestra librería (recordemos que en dicho paquete *está la* clase `Lienzo2D` definida en la práctica 7)[2].

Puesto que un `LienzoImagen2D` ha de permitir dibujar sobre él (al igual que lo hacía `Lienzo2D`), pero además ha de incorporar la posibilidad de mostrar una imagen, se recomienda definir la nueva clase `LienzoImagen2D` heredando de la clase `Lienzo2D`. De esta forma, y gracias a la herencia, se gestionará todo lo relativo al dibujo de formas y sus atributos sin necesidad de incorporar nuevo código[3]. Así, la nueva clase se centrará sólo en lo relativo a la imagen:

---

[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] 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`)

[3] Una mala praxis sería editar la clase `Lienzo2D` ya existente y añadirle nuevas propiedades y métodos.

- Habrá una variable de tipo `BufferedImage` que almacenará la imagen vinculada al panel. 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, en la nueva clase `LienzoImagen2D`, en lugar del método `paint` se recomienda sobrecargar el método[4,5] `paintComponent`:

```
public void paintComponent(Graphics g){
  super.paintComponent(g);
  if(img!=null) g.drawImage(img,0,0,this);
}
```

- 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 `LienzoImagen2D`. 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);
}
```

---

[4] La implementación por defecto del método `paint` de la clase `JComponent` llama, en este orden, a los métodos `paintComponent`, `paintBorder` y `paintChildren`. El primero de ellos es el que contiene el código que determina cómo se pinta el componente (los otros dos pintan el borde y llaman recursivamente al `paint` de los componentes/contenedores hijos). De hecho, para aquellas clases que heredan de contenedores/componentes Swing, la literatura aconseja sobrecargar el método `paintComponent` y no el `paint` (aunque en algunos casos puede interesar "jugar" con ambos, como ocurre en esta práctica). Para más detalles, se aconseja leer el artículo "Painting in AWT and Swing".

[5] Nótese que la sobrecarga del método `paint` dada por:

```
public void paint(Graphics g){
  super.paint(g);
  if(img!=null) g.drawImage(img,0,0,this);
}
```

no daría el resultado esperado porque la imagen se superpondría a lo previamente dibujado (en este caso, a las formas dibujadas en el método `paint` de la clase Lienzo2D).

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[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 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) {
    JFileChooser dlg = new JFileChooser();
    int resp = dlg.showSaveDialog(this);
    if (resp == JFileChooser.APPROVE_OPTION) {
    try {
      BufferedImage img = vi.getLienzo().getImage();
      if (img != null) {
        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 `LienzoImagen2D`:

[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.

```
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, incorporar en el método anterior el código necesario para crear una nueva imagen que en la que se dibuje la imagen del lienzo más las formas que éste incluye. Una vez implementado, en el método `menuGuardarActionPerformed` sustituiríamos la llamada `getImage()` por `getImage(true)`.

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[7].
- 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[8].
- 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 de cara a próximas prácticas (téngase en cuenta que iremos ampliando nuestro entorno gráfico, incluyendo nuevas barras y funcionalidades, por lo que se aconseja hacer un diseño sencillo y práctico). Se recomienda un diseño en línea similar al siguiente[9]:

- Incluir descripciones emergentes ("*tooltips*")
- Cuando se cambie de una ventana interna a otra, hacer que los botones de forma y atributos de la ventana principal se activen conforme a la forma y atributos del correspondiente lienzo.
- Mostrar en la barra de estado las coordenadas del puntero al desplazarse sobre el lienzo[10].

---

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

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

[9] En la web se incluyen los iconos mostrados en la figura. Para el caso de la lista desplegable, se aconseja leer cómo hacer celdas personalizadas mediante ListCellRender.

[10] La gestión de los eventos de movimiento de ratón no se hará desde la clase Lienzo2D (que en este caso es el generador), sino desde la ventana principal (o la interna).