PRÁCTICA 7

Ejercicio “Paint Básico 2D”

Descripción del ejercicio

El objetivo de esta práctica es realizar una versión mejorada del “Paint Básico” realizado en la práctica 4. En este caso, usaremos `Graphics2D` e incluiremos las siguientes nuevas funcionalidades:

- Entorno multiventana (i.e., la ventana principal tendrá un escritorio con ventanas internas, cada una de ella con su propio lienzo de dibujo).
- El lienzo mantendrá todas las figuras que se vayan dibujando
- Desplazamiento de las figuras previamente pintadas
- Grosor del trazo
- Transparencia
- Alisado de las formas

El aspecto visual de la aplicación será el mostrado en la Figura 1. En el escritorio se podrán tener tantas ventanas internas como se quiera, cada una de ellas con su propio lienzo de dibujo.

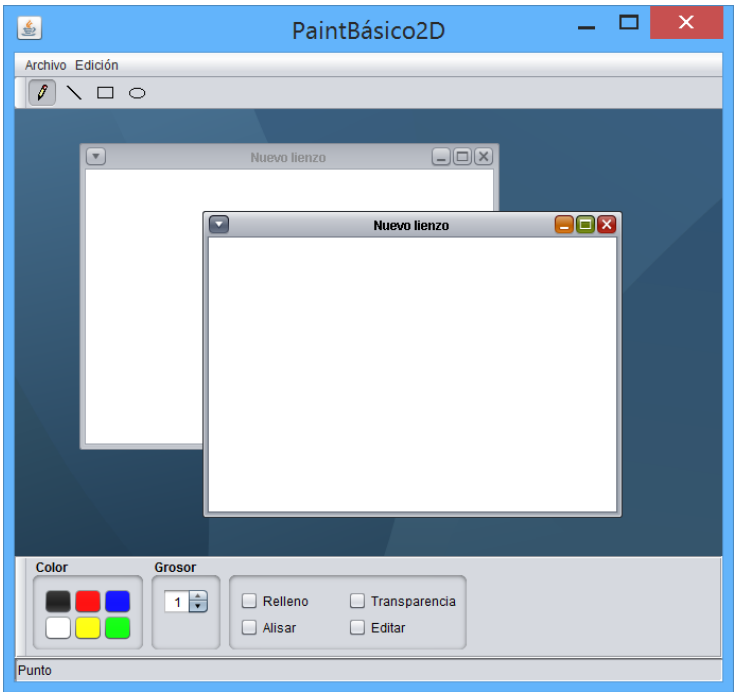


Figura 1: Aspecto de la aplicación

Al igual que en la práctica 4, el usuario podrá elegir entre cuatro formas de dibujo: **punto, línea, rectángulo y elipse**, que seleccionará mediante botones situados en la barra de herramientas (la barra de estado mostrará la forma activa). Además, habrá otra barra de herramientas (`JToolBar`) en la parte inferior para poder seleccionar los atributos de dibujo; concretamente, se podrá elegir entre un conjunto de **colores** predeterminados, el **grosor** del trazo (seleccionable mediante un *spinner*), si la forma está o no **rellena**, si se aplica o no **transparencia**[1]

[1] Entendida como semitransparencia correspondiente a un alfa 0.5

y si se **alisan** o no las formas en el renderizado (asociadas a estas tres últimas opciones, incluiremos tres casillas de validación).

El lienzo de cada ventana interna mostrará **el conjunto de formas dibujadas** en ese lienzo (en este caso, y al contrario que en la práctica 4, hay que mantener las figuras dibujadas usando un vector de formas). Todas las formas[2] se mostrarán con los mismos atributos de color, grosor, relleno, transparencia y alisado (seleccionados en el panel inferior), de forma que un cambio en un atributo implicará el cambio de dicha propiedad en todas las formas dibujadas[3].

El usuario podrá **mover las figuras** que ya estén dibujadas. Para ello, se incluirá en la ventana principal una casilla de validación (`JCheckBox`), de forma que, si está seleccionada, indicará que la interacción del usuario será para mover las figuras (en caso contrario, dicha interacción implicará la incorporación de nuevas 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 crear una nueva ventana interna con un lienzo vacío (sobre el que se podrá empezar a dibujar), mientras que "Abrir" y "Guardar" deberán lanzar el diálogo correspondiente (si bien no se abrirán ni guardarán archivos). El menú edición tendrá tres opciones: "Ver barra de estado", "Ver barra de formas" y "Ver barra de atributos" que activen/desactiven respectivamente la barra de estado, la de formas y la de atributos.

Algunas recomendaciones

1. Crear una biblioteca propia que contenga paquetes en los que incluir las clases de nueva creación que puedan ser necesarias en futuras prácticas. En particular, se recomienda incluir en esa biblioteca la clase correspondiente al lienzo y, si las hubiese, las clases de creación propia correspondientes a formas. Para ello:

    - Crear un proyecto en NetBeans de tipo "*Java Class Library*" de título *SM.XXX.Biblioteca*, con "XXX" las iniciales del estudiante

    - En el proyecto, dentro de la carpeta "Paquetes de fuentes", se crearán tantas subcarpetas como paquetes tenga nuestra biblioteca (en el menú contextual del proyecto, seleccionamos *Nuevo → Java Package*). En particular, se propone crear dos paquetes:

        - *sm.xxx.iu*, en la que se irán incluyendo componente y contendores de propósito general que puedan ser útiles a la hora de diseñar interfaces de usuario (por ejemplo la clase `Lienzo2D` que comentaremos posteriormente).

        - *sm.xxx.graficos*, en la que se incluirán clases de diseño propio relativas a gráficos (por ejemplo, aquellas correspondientes a clases de formas).

    - Una vez creada la biblioteca, hay que recordar incluirla en aquellos proyectos en los que vaya a utilizarse (a través de *Propiedades → Biblioteca → Añadir proyecto*). En particular, hay que añadirla en el proyecto de esta práctica 7.

[2] Se puede considerar que aplicamos la misma propiedad a todos los lienzos (en cuyo caso sería una propiedad de la clase), o distinta para cada lienzo (en cuyo caso sería una propiedad de la instancia). La primera opción es más sencilla de implementar, si bien se aconseja la segunda.

[3] Esto simplifica notablemente la aplicación, ya que, si se considerase como requisito que cada forma mantuviese los atributos con los que se dibujó, implicaría la necesidad de nuevas clases que agruparan información geométrica y de atributos (recordemos con los objetos `Shape` sólo contienen datos geométricos).

2. Ya centrados en el proyecto NetBeans propio de la práctica 7, y para realizar un entorno multiventana, aplicaremos lo ya visto en la práctica 3:

   - Anadir un escritorio en el centro de la ventana principal (donde se tuviese el lienzo en la práctica 4). Para ello, incorporar un `JDesktopPane` usando el NetBeans (área "contenedores swing").
   - Crear una clase propia *`VentanaInterna`* que herede de `JInternalFrame` (para ello, usar NetBeans). Activar las propiedades "*closable*", "*iconifiable*", "*maximizable*" y "*resizable*".
   - Para incorporar una ventana interna, 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 (*`vi.lienzo`*) y a sus métodos y variables.

3. Al igual que en la práctica 4, para el área de dibujo se recomienda crear una clase propia `Lienzo2D` que herede de `JPanel` (que incluiremos en la librería que hemos creado).En este caso, el lienzo estará situado dentro de la ventana interna (y no en la principal). Dicha clase gestionará todo lo relativo al dibujo: vector de formas, atributos, método `paint`, gestión de eventos de ratón vinculados al proceso de dibujo, etc. En esta clase:

   - En el método `paint` debería contener código centrado sólo en el dibujo de formas (mediante llamadas a métodos *`draw`* y *`fill`*), no de creación de objetos *`Shape`*. Por ejemplo:

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

   - Se recomienda definir métodos *`createShape`* y *`updateShape`* para la creación y modificación de formas; será dentro de dichos métodos donde se considerarán las diferentes casuísticas en función de la forma[4]. Estos métodos serán llamados

[4] En la práctica 5 no hacía falta distinguir entre unas formas u otras ya que sólo creábamos rectángulos (r=*`new Rectangle(…)`*) y los modificábamos (*`r.setFrameFromDiagonal(…)`*); ahora, el "*new*" dependerá de la forma a crear y los métodos para modificar de la clase correspondiente a la forma. Por ello se recomienda definir dos métodos *`createShape`* y *`updateShape`* que consideren las diferentes casuísticas en función de la forma (en un caso pare crear, en el otro para modificar); en particular, las sentencias *`switch`* (o *`if-else`*) que en la práctica 4 estaban localizadas en el método *`paint`*, ahora se encontrarán en este tipo de métodos.

desde los manejadores de eventos: el de creación desde el *mousePressed* y el de actualización desde el *mouseDragged* y *mouseReleased*.

- Al igual que en la práctica 5, se recomienda definir un método *getSelectedShape* para gestionar la selección de una forma. En dicho método se comprobará si el punto donde se ha hecho el clic está contenido dentro de alguna de las formas del lienzo[5].

- Para desplazar una forma, ya vimos en la práctica 5 que no existe un método en la clase *Shape* común para todas las formas (tipo *setLocation*); de hecho, en muchas clases no está definido un método específico que permita reubicar la forma (p.e., en la práctica 5 se explicó el caso de la *Line2D*) y hay que acudir a otro tipo de métodos (que dependen de la clase). Por ello, para esta práctica se recomienda definir un método *setLocation(Shape s, Point2D pos)* que considere las diferentes casuísticas en función de la forma[6].

Entrega del ejercicio

La entrega se hará en dos fases:

1. 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)
2. 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 NetBeans) y el ejecutable[7] (.jar).

Para la entrega final, además, habrá que incluir un documento PDF que responda a estas dos cuestiones:

a. Si la práctica exigiese que cada forma mantuviese los atributos originales (color, grosor relleno, etc.) con los que se dibujó, ¿serviría el enfoque actual de tu práctica? En caso negativo, ¿de qué forma habría que reenfocarla?

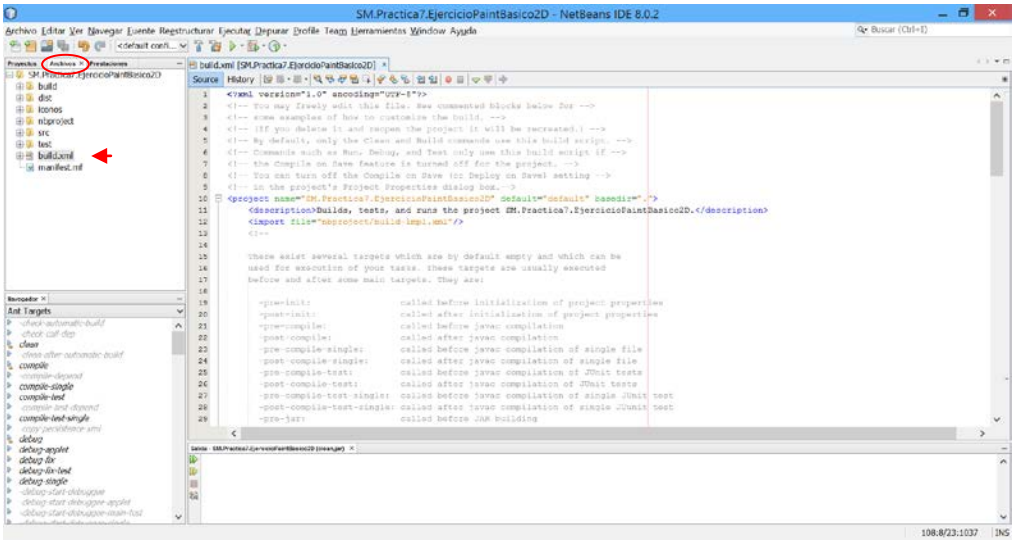
b. Al margen de la cuestión anterior, y centrándonos exclusivamente en los aspectos geométricos, ¿qué opinas sobre el diseño de las clases *Shape*? ¿son sus métodos adecuados o introducirías mejoras?

[5] En el caso de líneas, no sería "contenido" sino "cerca de". Para este caso, ha de considerarse lo explicado en la práctica 5.
[6] La solución correcta, desde el punto de vista del diseño, sería la creación de clases propias que solventaran las carencias de la clase *Shape* y subclases (como se apuntaba en la práctica 5 para el caso de la línea).
[7] Véase apéndice.

Apéndice: Incluir todas las bibliotecas en un único JAR

Como sabemos, para un proyecto dado, NetBeans genera el correspondiente fichero *.jar* en la carpeta */dist*. No obstante, si el proyecto usa bibliotecas externas (como ocurre en esta práctica), éstas no se incluyen en el fichero *.jar*, sino que, en su lugar, crea una carpeta */lib* (dentro de */dist*) en la que incorpora todas estas bibliotecas. Esto implica que, para ejecutar el fichero *.jar*, tiene que estar siempre presente la carpeta[8] */lib* (haciendo más "engorrosa" la posible distribución de nuestro programa -véase el fichero *README.TXT* generado por NetBeans-). Si quisiéramos generar un *.jar* que empaquetase todas las bibliotecas en un único fichero, habría que hacer lo siguiente:

1. Nos vamos a la sección "Archivos" (junto a "Proyectos", en el panel superior izquierdo) y seleccionamos el fichero *build.xml*:



   Como vemos, hay muy poco código (la mayoría del fichero es una sección comentada que explica cómo ampliarlo)

2. Incorporamos el siguiente código XML al final del archivo *build.xml* (antes del *tag* de cierre *</project>*):

```
<target name="-post-jar">
  <property name="pack.jar" value="dist/${application.title}.pack.jar"/>
  <echo message="Packaging into a single JAR at ${pack.jar}"/>
  <jar jarfile="${pack.jar}">
    <zipfileset src="${dist.jar}" excludes="META-INF/*" />
    <zipgroupfileset dir="dist/lib" includes="*.jar" excludes="META-INF/*"/>
    <manifest>
      <attribute name="Main-Class" value="${main.class}"/>
    </manifest>
  </jar>
</target>
```

3. Ahora, al "Limpiar y construir" nuestro proyecto, además del *.jar* que se obtenía antes, se generará otro fichero *.pack.jar* que incluirá todas las bibliotecas y, por lo tanto, se podrá ejecutar de forma autónoma sin necesidad de estar junto a la carpeta */lib*. En caso de tener que distribuir vuestro programa, usad este fichero.

[8] Si no está la carpeta */lib*, no tendrá acceso a las clases de las bibliotecas y lanzará excepciones cuando trate de usarlas. Para poder ver el trazado de estas excepciones, habrá que ejecutar nuestra aplicación desde una ventana de comandos. Por este motivo, se aconseja que antes de distribuir una aplicación a un usuario final, ésta se ejecute desde una ventana de comandos para asegurarnos que no se lanzan excepciones de inicio.