Trabajo Práctico Especial
Programación Orientada a Objetos

Alumnos:

- Paula Andrea Domingues 60148
- Pedro Jeremías López Guzmán 60711

En este informe se detallarán las modificaciones realizadas a la implementación dada por la cátedra, así como las funcionalidades agregadas a la aplicación gráfica para el Trabajo Práctico Especial de Programación Orientada a Objetos.

Funcionalidades Agregadas:

1. Creación de nuevas figuras:

   Se agregó a las lista de figuras provista por la cátedra la posibilidad de dibujar Elipses, Cuadrados y Líneas.

2. Selección Múltiple:

   Al presionar el mouse en un punto del canvas y arrastrarlo hacia abajo y a la derecha y soltarlo, todas las figuras que estén complementamente contenidas en el rectángulo formado por ambos puntos, se seleccionarán.

3. Borrado:

   Se agregó un nuevo botón en el panel izquierdo, cuya funcionalidad es eliminar las figuras seleccionadas.

4. Borde y Relleno:

   Se agregaron a la barra izquierda dos nuevas funcionalidades, borde y relleno. La funcionalidad borde permite elegir el grosor y color del borde de la figura y relleno permite elegir el color interior de la misma. Estas propiedades son aplicables tanto posterior como previamente a haber dibujado la figura. A su vez, también es posible cambiar estas propiedades en el caso que hubieran múltiples figuras seleccionadas.

5. Profundidad:

   Se agregaron dos botones en la barra izquierda para manejar el orden en el que se dibujan las figuras. El primer botón “Al frente” mueve las figuras seleccionadas, manteniendo el orden relativo entre ellas, para que estén por encima de las demás. El segundo botón “Al fondo” mueve a las figuras seleccionadas para que estén por debajo del resto.

Cambios a la implementación provista por la Cátedra:

En esta sección se describirán aquellos cambios realizados al código ya implementado por la cátedra, y se explicará la razón de los mismos.

Al abrir por primera vez los archivos, se pudo observar la separación hecha entre frontend y backend, y se decidió mantener dicha estructura. Luego, posterior a la creación de las nuevas clases de figuras, se modificó la herencia de la clase *Circle* para que heredara el comportamiento de la clase *Ellipse*, ya que gran parte de este era similar. Esto se llevó a cabo también con *Square* y *Rectangle*.

Asimismo, dentro de aquellas implementaciones que podían reemplazarse por otras que se ajusten más al Paradigma Orientado a Objetos, se encontraron varios métodos que presentaban validaciones para asegurarse que una variable, llámese var, no fuera nula y tuviera un valor asignado mediante *var == null*. Las mismas fueron reemplazadas por *Optional.ofNullable(var).isPresent()*, a modo de lograr una mayor adaptación al POO y, a su vez, lograr una mayor claridad en el código.

También se cambiaron los scopes de algunas variables de algunas clases para que la visibilidad de las mismas sean privadas o protegidas.

Luego, en la clase *PaintPane* una modificación fue la función *figureBelongs*, cuya funcionalidad era detectar si un punto pertenecía a una figura en particular. Teniendo en cuenta el paradigma orientado a objetos, se consideró que la correcta implementación sería que esta función sea un método de instancia de la clase abstracta *Figure* en el paquete de backend. Así fue creado el método *containsPoint*, que recibe un *Point* y retorna verdadero si las coordenadas de dicho *Point* están dentro de la figura, y falso si están afuera.

Dentro de la clase *PaintPane* también se modificó la funcionalidad del movimiento de las figuras. Primero se retocó el algoritmo para que el movimiento sea más cómodo y familiar con otras aplicaciones gráficas. Luego se tomó la decisión de que cada figura posea un método *move* por el cual a través de dos parámetros que representan la distancia trasladada en los ejes X e Y, la misma modifique sus coordenadas. Otro cambio fue el de la variable *statusPane* ya que esta fue dada como variable de instancia, pero sólo es usada en el constructor, por lo que se eliminó dicha variable y se utilizó el parámetro *statusPane* para cumplir su funcionalidad.

En cuanto a la clase *StatusPane*, se agregaron los métodos *updateOrDefault, showStatus* y *showStatusNewFigure,* que agregan las funcionalidades de actualizar el Label con un string, o un valor por defecto si el string está vacío; mostrar las figuras seleccionadas y mostrar la creación de una figura nueva respectivamente.

Otro cambio a destacar es la creación de un Enum llamado *FigureTogglesEnum* para los *ToggleButton* que se encargaban de crear figuras, ya que tenían un comportamiento similar que podía ser encapsulado en una clase. De esta forma, la sección de código correspondiente al comportamiento del mouse en el canvas se reescribió de una forma mucho más comprensible.

Por último, se realizaron varias modificaciones dentro del *CanvasState*. Luego de la creación de la interfaz *Drawable*, explicada a continuación, la lista deja de ser de *Figures* para convertirse en una de elementos de tipo *Drawable*. A su vez, se decidió que sería más conveniente al momento de llevar a cabo determinadas funcionalidades que la lista fuera una LinkedList en vez de la ArrayList originalmente implementada. Esto permite que el llevar las figuras al frente y al fondo se convierta en una tarea más sencilla, y sin la necesidad de recorrer toda la lista para insertar en el lugar requerido.

Modificaciones realizadas al Proyecto Original:

En esta sección se describirán aquellos métodos que fueron agregados durante el proyecto que permiten que las funcionalidades se ejecuten de forma eficiente y permitan su modularización, pero que no modifican el código provisto por la Cátedra.

Dentro del backend, se agregó el metodo *isContained*, de modo que, al momento de realizar la selección de múltiples figuras, cada una ellas sepa si está contenida o no dentro del rectángulo imaginario de selección que se le pasa como parámetro. También el método privado *isBetween*, particular de la clase Line. Este método recibe tres puntos, llámese puntoA, puntoB y puntoC y es utilizado por *isContained* para determinar si el puntoC se encuentra entre puntoA y puntoB. Este método, si bien podría simplemente ser declarado dentro de *isContained*, fue agregado para buscar mayor claridad al momento de leer el código. Ambos métodos devuelven un booleano.

Teniendo en cuenta el paradigma se tomó la decisión de que cada objeto que debía ser dibujado en el canvas posea un método de instancia para poder encargarse de su renderizado. El implementar esta funcionalidad en las clases de figuras hubiese sido erróneo, ya que se estaría rompiendo la separación entre backend y frontend. La decisión tomada fue crear una interfaz llamada *Drawable* cuya principal funcionalidad sería la del método “draw”, que recibiría como parámetro un *GraphicContext* en el cual el objeto debía ser dibujado. Así se crearon una nueva rama de clases para el manejo de figuras. Cada clase hereda el comportamiento de una figura en particular, mientras que implementa dicha interfaz, definiendo dicho método. De esta manera se modificó la clase *CanvasState* para que posea una lista de *Drawable* en vez de *Figure*, así también como la clase *PaintPane* y su función *redrawCanvas*.

A continuación se detallan los demás métodos de la interfaz Drawable:

- *isContained*: método que evalúa si el Drawable está dentro de un rectángulo. Necesario para la funcionalidad de selección múltiple.
- *move*: método para mover el Drawable.
- *containsPoint*: método que evalúa si un punto esta dentro de un Drawable.

- *setFillColor* & *getFillColor*: Métodos que setean y devuelven el color de relleno del drawable, respectivamente.
- *setStrokeColor* & *getStrokeColor*: Métodos que setean y devuelven el color del borde del drawable, respectivamente.
- *setStrokeWidth* & *getStrokeWidth*: Métodos que setean y devuelven el grosor del borde del drawable, respectivamente.

Tanto *isContained*, *move* y *containsPoint* están implementadas en la clase *Figures* en el back end, pero las contiene también esta interfaz en el front para poder utilizarlas después en el Paint Pane.

Es pertinente realizar una breve descripción del Enum *figureTogglesEnum*. Este cuenta con un método abstracto *newFigure*, el cual es utilizado al momento de crear cada figura en el canvas, dependiendo del botón que esté seleccionado, realizando previamente las validaciones necesarias respecto de las coordenadas. A su vez, cuenta con un String, el cual es utilizado para darle el nombre al ToggleButton que se crea con cada elemento del mismo. Por último, el método *matchAndGetButtonName*, que recibe un String de nombre y devuelve el elemento del *figuresTogglesEnum* que lo contiene, es utilizado para lograr mayor claridad en el código al crear una nueva figura.

A su vez, se creó la clase abstracta *regularButtons*, que extiende de Buttons y de la cual extienden *deleteButton*, *toFrontButton* y *toBackButton*. La intención de la creación de estas clases es lograr que las mismas tengan los mismos métodos que un Button de JavaFx, y otros adicionales que se le son agregados.

Por último, es preciso destacar la creación de la clase *closeProcesses* y la interfaz *PointValidator*. Esta clase se ocupa de lanzar la alerta correspondiente al momento de cerrar la aplicación, tanto como para cuando se desee realizar mediante el menú o por medio del botón “exit”. Por su parte la interfaz asegura la funcionalidad de validar si los puntos dados para la creación de un *Drawable* son válidos. Los botones en *FigureTogglesEnum* implementan dicha interfaz para no generar errores a la hora de crear figuras.

Problemas encontrados durante el desarrollo:

El primer problema fue el uso de las aplicaciones gráficas y botones de la librería JavaFx. Para ello se investigaron de la documentación de la API de Java las clases y funcionalidades necesarias.

Otro problema que se tuvo fue el mantener la separación entre Backend y Frontend. Para evitar un desarrollo imperativo se necesitaba que cada figura sea responsable de su dibujo, pero implementar un método con este objetivo en las clases de figuras rompía con la estructura del programa. El problema se solucionó con la implementación de la interfaz *Drawable*.

La implementación de los botones en la clase *PaintPane* no seguía con el paradigma orientado a objetos, por lo que se decidió crear clases individuales para

cada botón. Surgió un problema al notar que el comportamiento de los botones cuya función era crear figuras era similar, y se necesitaría encapsular dicho comportamiento para no repetir código. La solución hallada fue crear el Enumeration llamado *figuresTogglesEnum* definiendo el comportamiento de cada botón de los antes mencionados.

Un problema producto a la decisión anterior fue que al necesitar uno de los botones de *figuresTogglesEnum* se utilizaba la función *valueOf*, la cual devuelve el botón cuyo identificador sea igual al parámetro pasado. Esto no permitía que los identificadores de los elementos del Enum fueran distintos del String name del mismo, por lo que no era permitido que los mismos estuviesen en mayúsculas (siguiendo un buen estilo), ni que pudiesen poseer tildes. La solución fue la creación de la función *matchAndGetButtonName* la cual devuelve el elemento del Enum cuyo atributo “name” coincida con el parámetro de la función.

Otro problema a comentar es el error en el método *isBetween* de la clase *Line*. En caso de que se dibujara una línea cuyos extremos fuesen el mismo punto, debido a como está implementada la función,siempre devolverá verdadero. La solución se desarrolló sobreescribiendo la función equals en la clase *Point*, para poder evaluar si los puntos extremos de una instancia de la clase *Line* estaban posicionados en el mismo punto. En caso de que eso suceda, se omite el proceso para calcular si el tercer punto pertenece a la línea y se evalúa si el tercer punto es igual al primero.

Por último, un problema que surgió una vez implementados todos los métodos correspondientes fue que, al seleccionar una figura y trasladar a esta de una ubicación en la esquina superior izquierda hacia la esquina inferior derecha, se creaba el rectángulo imaginario de selección múltiple, por lo que al finalizar de trasladar a la antedicha se encontraban otras figuras seleccionadas. Esto fue rápidamente solucionado mediante validaciones en la clase *PaintPane*.

Comentarios Adicionales:

Se considera pertinente destacar que a medida que el proyecto fue avanzando ciertas clases, específicamente *Circle* y *Square*, quedaron en desuso, dado que los comportamientos de las mismas podían ser heredados. Sin embargo, se decidió que, a pesar de haber quedado obsoletas, estas se mantuvieran en el diagrama de clases para procurar un mayor entendimiento del desarrollo del práctico.

Para finalizar, dentro del zip se encuentra una carpeta llamada “*cursores*” la cual cuenta con varias imágenes .png. Se pueden observar en varias ocasiones a lo largo del código métodos setCursor, los cuales utilizan estas imágenes para cambiar el estilo del cursor. Para disfrutar de una implementación completa, se recomienda tener a esta en el mismo directorio en el que se encuentre la carpeta *src*.

Este adicional surge meramente de la investigación respecto de las distintas funcionalidades que se podían llevar a cabo utilizando la herramienta JavaFx.