Informe – Trabajo Practico Especial

Programación Orientada a Objetos (72.33)

Integrantes:

Boullosa, Juan Cruz. Legajo: 63414.

Deyheralde, Ben. Legajo: 63559.

Mutz, Matías. Legajo: 63590.

Fecha de Entrega:

11/12/2022

Cambios al código provisto

Para comenzar, hicimos qué Circle y Square extiendan a Ellipse y Rectangle respectivamente, para aprovechar el comportamiento ya que un cuadrado es un rectángulo y un círculo una elipse. Siendo este, un cambio de estilo para aprovechar la herencia.

Con el fin de mantener el encapsulamiento propio del estilo de la programación orientada objetos, decidimos convertir a la mayor cantidad de variables y métodos en private tanto en las clases del frontend, como también del backend. En particular, vale la pena destacar que, debido a esto, agregamos los métodos moveX, moveY a la clase Point, métodos que son utilizados para mover el punto en cuestión.

A continuación, nos encontramos con que en la parte del frontend, en particular en PaintPane, estaba mezclado el frontend con el backend. No solo eso, sino que, debido a mal estilo, había una gran cantidad de if anidados, lo cual hacía que esta clase tuviese métodos muy imperativos los cuales van en contra del paradigma. Es por esto por lo que decidimos crear una gran cantidad de métodos en canvasState ya que el mismo pertenece al backend. Dichos métodos modifican el estado del programa y permiten que el frontend simplemente solicite dichos cambios con llamados a las distintas funciones, para luego visualizarlos.

En particular, para solucionar el problema de una gran cantidad de ifs decidimos crear una clase abstracta SpecialButton que extiende de ToggleButton y que a su vez es padre de la clase de squareButton, rectangleButton, ellipseButton y circleButton. Esta modificación fue hecha para mejorar el estilo y la eficiencia. A su vez, creamos otra clase abstracta, ClickableButton, que también extiende de ToggleButton y cuyos hijos son SelectionButton y CopyFormatButton, dichas clases tienen un método, actonClick, que determina qué cambios se realizar si es que se clickea sobre una figura.

En el archivo AppMenuBar cambiamos el método showAndWait que tiene alert por una cuestión de estilo.

Agregamos setters y getters en las clases correspondientes, esto debido a que como previamente mencionamos, por cuestiones de estilo, optamos por convertir la mayor cantidad posible de variables a private.

Funcionalidades agregadas

La primera funcionalidad nueva es la de darle un formato a las figuras, creamos una clase Format, la cual contiene el formato siendo este el color del relleno y el del borde de la figura, como también también el grosor de dicho borde almacenados en variables privadas. Como todas las figuras tienen formato, decidimos convertir la interfaz Figure en una clase abstracta, que almacena el formato.

Además, para setear y modificar el formato de las figuras agregamos en PaintPane, dos ColorPicker y un Slider de los cuales se extrae la información del formato.

La próxima funcionalidad desarrollada es la de copiar, cortar y pegar. Para ello, creamos una clase CutCopyPastePane, que extiende a BorderPane y contiene los botones de copiar, cortar y pegar. A su vez se envía la referencia de dichos botones a PaintPane, para que se realicen las acciones al apretar el botón correspondiente o utilizar el shortcut ctrl+C, ctrl+X, ctrl+V para activar cada botón.

Por último, desarrollamos la tercera funcionalidad la cual es deshacer o rehacer cambios. Para ello, tuvimos que desarrollar varias cosas.

- Por un lado, creamos una clase Change, la cual representa un cambio. En dicha clase se almacena una copia de la figura antes del cambio, una copia de la figura después del cambio y un elemento de Enum Action, el cual indica cuál es la acción realizada.
- El enum Action tiene todos los cambios posibles y entre sus métodos, está undo, redo y getMessage con los primeros dos métodos realizan los cambios correspondientes, mientras que el último devuelve el mensaje que debe aparecer al lado de los botones de rehacer y deshacer.
- A su vez para poder rehacer y deshacer los cambios, creamos 2 LinkedList que contienen Change. Una tiene los cambios que se hicieron y otra que contiene los cambios desechos.

Por lo tanto, todo el backend se organiza a través de CanvasState, quien a su vez le brinda información al Frontend. Motivo por el cual, en CanvasState hay métodos que permiten agregar las Figura a la lista de las figuras que aparecen en el Canvas, también tenemos otro para agregarlo a la LinkedList de los cambios realizados como también a la de los cambios deshechos. De la misma manera, disponemos de métodos para determinar la cantidad de cambios disponibles para deshacer, como también rehacer. A su vez, armamos métodos para realizar cada uno de los cambios posibles, como también mover una figura.

Por otro lado, al realizar una acción en la aplicación, en la mayoría de los casos, desde PaintPane, se llama a uno de los métodos del CanvasState, para que realice las modificaciones de estado correspondientes. Permitiendo que se reflejen los cambios en la interfaz gráfica.

Problemas encontrados

Uno de los problemas encontrados fue que cuando quisimos hacer los botones con imágenes en el costado, como lo son el de Copiar, Pegar y otros, nos encontramos con que el código provisto para hacer los botones no nos funcionaba. Después de un tiempo nos dimos

cuenta de que era porque estábamos usando Java 19 y ese código funcionaba en Java 8, así que cambiamos la versión a Java 8.

Otro problema fue que, al no tener implementado el equals en las figuras, se nos generaba un bug cuando movíamos una figura.

Debido a que el UML del trabajo implica conexión entre muchas clases y al querer ponerlo en este archivo en distintas imágenes queda muy desprolijo e inentendible, decidimos adjuntar la imagen de este en la carpeta comprimida que fue entregada.

Otra cosa que nos llevo más tiempo de lo esperado fue lograr que los botones queden centrados y que los Pane queden en el lugar que queríamos que estén. Suponemos que esto fue por no tener experiencia alguna con JavaFX pero pudimos ir solucionándolo sin mayores problemas.