Trabajo Práctico Especial

Instituto Tecnológico de Buenos Aires - Programación Orientada a Objetos (73.22)

Grupo 1

Ignacio Searles
isearles@itba.edu.ar
64.536

Augusto Barthelemy Solá
abarthelemysola@itba.edu.ar
64.502

Nicolás Arancibia Carabajal
narancibiacarabajal@itba.edu.ar
64.481

Tomás Borda
tborda@itba.edu.ar
64.517

10 de diciembre de 2023

Introducción

En el presente informe se trata el desarrollo de una aplicación de dibujo de figuras a partir de una implementación dada. En el mismo, se detallan modificaciones realizadas a la implementación inicial, dificultades halladas y sus respectivas resoluciones.

Desarrollo

A partir de la implementación parcial entregada por la cátedra, se realizaron diversas modificaciones con diversos fines. En primer lugar se corrigieron aspectos esenciales que violaban los principios del paradigma, tal como cambiar la visibilidad de los atributos de la clase Point de público a privado, de tal forma que no puedan ser accedidos y modificados desde cualquier lado más que desde la propia clase, lo mismo con la clase *Ellipse* de protegido a privado. Así también, se reestructuró el diagrama de clases teniendo en cuenta que había figuras que eran un caso particular de otras, por ejemplo el cuadrado es un rectángulo con todos sus lados iguales y el círculo una elipse con sus ejes equivalentes. Teniendo esto en mente, la clase *Square* debería extender de *Rectangle*, lo mismo con *Circle* y *Ellipse*, pasando en el constructor los parámetros correspondientes al caso particular. Por otro lado, la clase *CanvasState*, considerando que únicamente utilizaba métodos implementados en ArrayList, fue modificada de tal forma que extienda de la clase ArrayList en vez de poseerla como atributo, pudiendo usar los métodos propios de ArrayList sin pasar por *CanvasState*. Así también, se consideró que el color debería ser una propiedad de la figura y podría, por ende, ser almacenado en el back-end. Para no utilizar la clase Color propia de JavaFX se utilizaron los colores como String mediante su código hexadecimal. Por último y

no menos importante, se reestructuró todo el panel principal de dibujo, principalmente para corregir la lógica imperativa empleada para dibujar, agregar figuras al canvas o saber si un punto pertenece a la figura, pero también para ayudar a la lectura del código, ya que gran parte del front se encontraba allí, cuando uno podría separarlo por clases para favorecer la modularización y legibilidad del mismo.

En cuanto a la lógica empleada se refiere, se consideró que verificar de qué tipo de figura se habla para saber cómo actuar, no era la mejor solución dentro del paradigma de la programación orientada a objetos; sino que, aprovechando las interfaces o clases superiores y herencias de métodos cada figura podría saber cómo dibujarse a sí misma, así como verificar cuándo un punto está dentro de la figura. Éste último método pudo ser agregado a la interfaz de figuras ya existente en el back-end. No obstante, el método para dibujar las figuras en el canvas es propio de JavaFX, por lo que no podía simplemente ser movido a la interfaz e implementado por cada una de las figuras, ya que el back-end debía poder ser utilizado por otros front-ends con sus propios métodos de dibujo. Consecuentemente, tuvo que pensarse una modificación del diagrama de clases a modo de resolver esto.

En un primer momento, se implementaron clases *Drawable* en el front-end que extendieran de las clases del back-end y que tuvieran los métodos de dibujo propios de JavaFX. De esta forma, se implementó una interfaz *DrawableFigure* con métodos para dibujar las figuras, así como clases de figuras *Drawable* que implementaran dicha interfaz y extendieran o bien de su respectiva figura del back-end, como en el caso de un *DrawableRectangle*, o de otra figura dibujable, como por ejemplo *DrawableSquare*, a fin de evitar repetir código y aprovechar la herencia. No obstante, comenzó a presentarse una de las grandes dificultades del trabajo práctico, así como una limitación del lenguaje, y es que no tener la posibilidad de heredar de múltiples clases hizo que el diagrama de clases tuviera que ser reestructurado muchas veces. Con la estructura planteada, el círculo y cuadrado, al no extender de su clase del back-end, debían reescribir el toString o bien extender de su clase y reimplimentar los métodos de dibujo de la elipse o el rectángulo. Ambos casos llevaban a repetición de código. Así también, para la implementación de los efectos (2da funcionalidad), consideramos necesario que cada figura tuviera valores booleanos que indicaran el encendido o apagado de los efectos, cosa que no podía ser agregada en una interfaz *DrawableFigure*, por lo que se repensó la estructura general. Al hacer *DrawableFigure* una clase abstracta para favorecer la implementación de los efectos y los tags, nuevamente se presentaba el problema de extender de *DrawableFigure* y no poder extender de una figura del back-end por falta de herencia múltiple. Se intentó solucionar esto mediante una composición, donde cada figura dibujable tenía en un atributo a su figura base. Sin embargo, seguía resultando poco adecuado desde el punto de vista teórico, siendo que cada figura dibujable debería extender de su figura base y no componer. En cuanto a otras dificultades, al momento de implementar los grupos para la primera funcionalidad comenzaron a hallarse problemas de tipos al usar generics dentro de colecciones de clases que debían extender de figuras e implementar funciones de dibujo, inconvenientes los cuales podían ser resueltos mediante casteos explícitos de clases que aplicaban wildcards, pero esto arruinaba parte de la legibilidad y estilo del código.

En vista de las dificultades presentadas, y luego de pensar diferentes jerarquías de clases, implementarlas y analizarlas, se terminó concluyendo que la mejor forma de estructurar las figuras sería con una clase abstracta *Figure* que posee todos los atributos y métodos necesarios para todas las funcionalidades a excepción del dibujo de la figura, implementado mediante la clase *DrawableManager*. Dicha clase debe estar en el front-end y, en este caso, se implementa tanto para el rectángulo como para la elipse. Luego, con dicha estructura se evita tener figuras dibujables con problemas por falta de herencia múltiple, así como tener que realizar casteos explícitos. En

el anexo de este informe, es posible ver dicha estructura, junto con otras clases en el front-end realizadas para mayor modularización y reducción del código en el panel de dibujo, así como también es posible ver *SelectionManager*, utilizado para las acciones a realizar con el grupo de figuras seleccionadas, y *FigureGroup* cuya implementación se explica a continuación. (Ver Anexo 1)

En cuanto a las funcionalidades agregadas, para implementar la selección múltiple se implementó una clase *FigureGroup* la cual agrupa figuras de manera tal que a lo largo de todo el programa se trabaja con grupos de figuras en lugar de figuras en sí. Esto quiere decir que cuando se dibuja una figura se crea un grupo propio para ella, cuando se agrupan figuras se crea un nuevo conjunto con las figuras seleccionadas, se agrega dicho grupo al canvas y se remueven todos los grupos seleccionados, y al desagrupar se vuelven a crear grupos individuales para cada una de las figuras del conjunto, removiendo este del canvas y agregando los nuevos grupos de figuras generados. Al funcionar de dicha manera, es posible agrupar dos o más grupos entre sí o figuras con grupos, puesto que debido al diseño siempre se trabaja con *FigureGroup*, sin importar la cantidad de figuras presentes en dicho grupo. Así también, es posible desagrupar más de un grupo seleccionado. Estas decisiones de diseño fueron tomadas con la intención de asimilar las funciones presentes en aplicaciones de dibujo. Además, cada uno de los métodos propios de las funcionalidades a implementar se encuentran presentes en el grupo de figuras y, al llamar a un método que debe aplicarse a todas las figuras del grupo, itera sobre el conjunto, aplicando dicho método a cada figura. Un inconveniente al desarrollar la selección múltiple fue considerar que al mover figuras no se esté dibujando un rectángulo de selección. Para solucionar esto, se agregó una verificación adicional para evitar poder dibujar dicho rectángulo al tener un grupo de figuras seleccionado, así como un pequeña tolerancia al movimiento del mouse, y así distinguir cuando se hace click con la intención de seleccionar de cuando se está arrastrando con la intención de mover figuras.

Para la segunda funcionalidad, se realizó por un lado un panel en el front-end con los botones de los efectos, donde se controla la interfaz y los diferentes estados que puede tomar el botón. Por otro lado, la tarea del dibujo de los efectos está presente en *DrawManager* mediante métodos que deben ser implementados por cada figura en el front-end, así como los getters sobre los estados de los diferentes efectos se encuentran implementados en la clase abstracta figura.

Para la implementación de la tercera funcionalidad, los métodos requeridos para cada acción se encuentran en la clase abstracta *Figure*, muchos de los cuales son abstractos y deben de ser implementados por cada una de las figuras en el back, es decir que tanto *Rectangle* como *Ellipse* sobreescriben dichos métodos, *Square* y *Circle* simplemente las heredan.

En cuanto a la posibilidad de agregar tags, se implementaron con un Set dentro de cada figura, de forma tal que no puedan haber tags repetidos. Así también, al ignorar espacios, filtrar por una etiqueta que solo contenga espacio es equivalente a filtrar por una cadena de texto vacía. La misma se agrega a la colección de tags de la figura al instanciar cualquiera y es el equivalente a "no tener etiquetas". Al agregar otros tags, la cadena vacía se sobrescribe y se reemplaza por las nuevas etiquetas añadidas. Por otro lado, al agrupar dos o más figuras se muestra en el panel de escritura la concatenación de las etiquetas de cada una de las figuras del grupo mediante un método en *FigureGroup* que itera por las figuras del grupo y se encarga de concatenar los tags. Así, hasta no presionar el botón guardar, las figuras del grupo pueden conservar sus etiquetas individuales, pero al filtrar por la de alguna figura del grupo todo el conjunto es visible (No solo dicha figura del grupo con el tag filtrado).

Conclusión

En conclusión, pese a las dificultades y limitaciones técnicas presentadas durante el desarrollo, y gracias al involucramiento del grupo en la estructura general del proyecto, es posible evidenciar que, mediante diversas modificaciones a la implementación inicial, fue posible llegar a una solución que es clara, funcional, y modulariza las diferentes tareas y paneles de la aplicación.