UNIVERSIDAD DE CONCEPCIÓN

FACULTAD DE INGENIERÍA

DEPARTAMENTO DE INGENIERÍA INFORMÁTICA Y CIENCIAS DE LA COMPUTACIÓN

Programación II

Pizarra/Editor de diagrama UML

Bianca Seminario

Fecha: 20 de Diciembre, 2023

Índice

INTRODUCCIÓN:

Este proyecto presenta un "Pizarra/Editor de diagrama UML" diseñado para simplificar la creación y edición de diagramas UML. Con la capacidad de trabajar en múltiples pizarras simultáneamente, los usuarios pueden cambiar entre ellas sin perder cambios gracias a un sistema de pestañas o menú intuitivo. La herramienta incluye modos de edición, como borrar y crear formas UML mediante arrastre del mouse, y ofrece la conveniencia de identificar y eliminar elementos de forma precisa. La gestión de archivos se realiza de manera eficiente, permitiendo guardar y cargar pizarras, con la función de guardado automático al cambiar entre ellas. Además, se destaca por su sencillez al permitir a los usuarios personalizar colores con botones directos en la interfaz gráfica.

ENUNCIADO: Pizarra/Editor de diagrama UML

Se debe crear una pizarra múltiple que permita dibujar trazos, rectángulos, entidades y conectores UML mediante arrastre del mouse. La pizarra debe contar con modos de edición, como borrar y crear las diferentes formas de UML. Para eliminar, se busca el elemento que contenga los píxeles encerrados por un rectángulo fantasma que se crea entre el clic del mouse y la liberación del mismo.

La pizarra múltiple (con un sistema de pestañas o un menú) consiste en tener una de ellas en el panel central en un momento dado, pudiendo cambiar de una a otra sin perder los cambios realizados. Se debe poder guardar la pizarra múltiple en un archivo y cargarla posteriormente. Además, se debe permitir borrar la pizarra completa. La pizarra múltiple se debe guardar automáticamente en el archivo al cambiar de una pizarra a otra. Por último, se debe poder elegir el color de las líneas y entidades mediante botones en la interfaz gráfica (GUI)

Prototipo Inicial de la interfaz

Se planea que al iniciar el programa, el usuario tendrá la opción de crear un nuevo archivo o abrir uno previamente guardado. Esta funcionalidad permitirá al usuario la flexibilidad de continuar con la edición de sus proyectos anteriores o empezar uno completamente nuevo desde cero.

El usuario dispondrá, en el lado izquierdo del panel, de una barra para seleccionar el tipo de flecha o clase que desea agregar a la pizarra UML. Además, en la parte inferior, encontrará una serie de botones para diversas funcionalidades, como eliminar todo, guardar la pizarra, entre otras opciones. Por otro lado, el usuario tendrá la capacidad de crear múltiples pestañas según sus necesidades y la opción de cargar un archivo previamente creado. En caso de no encontrar el archivo o no poder abrirlo, se mostrará un mensaje de advertencia.

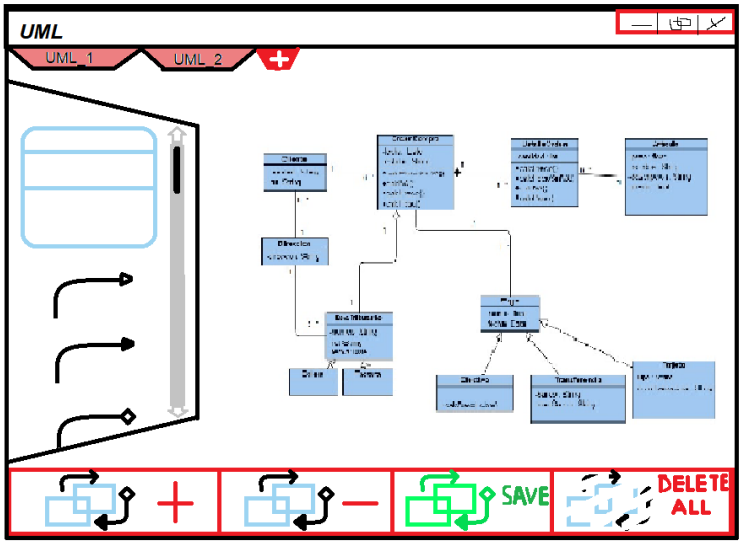
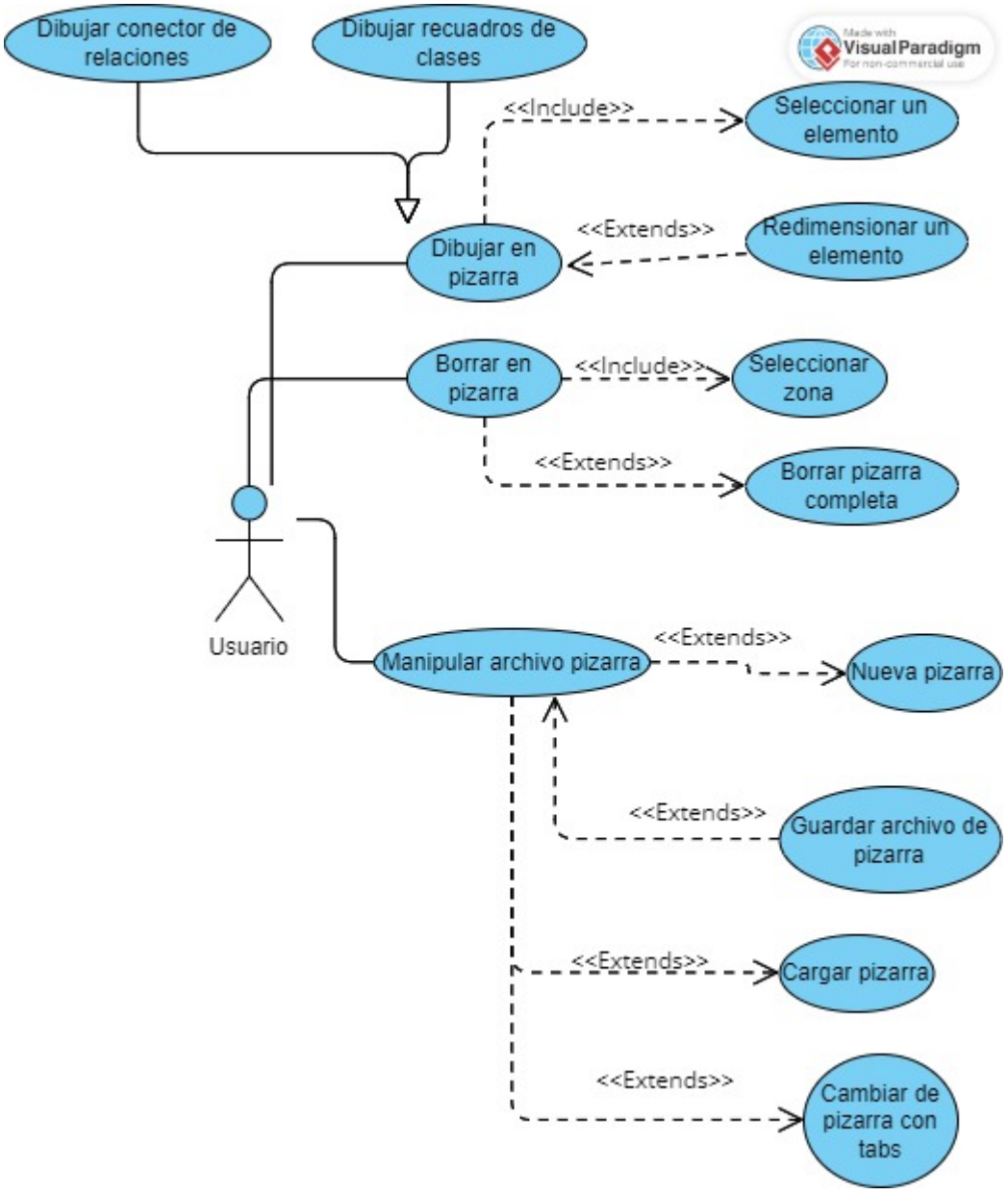
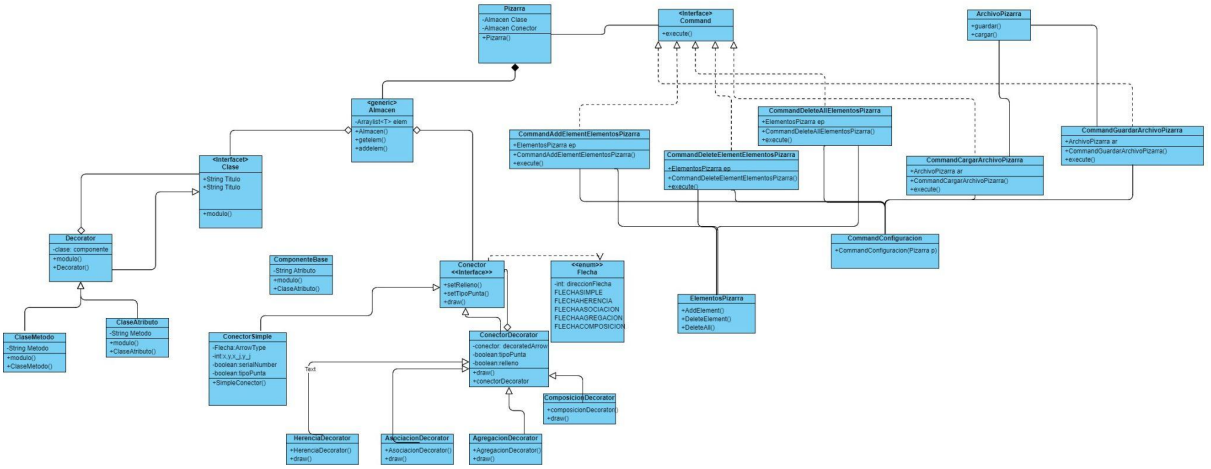


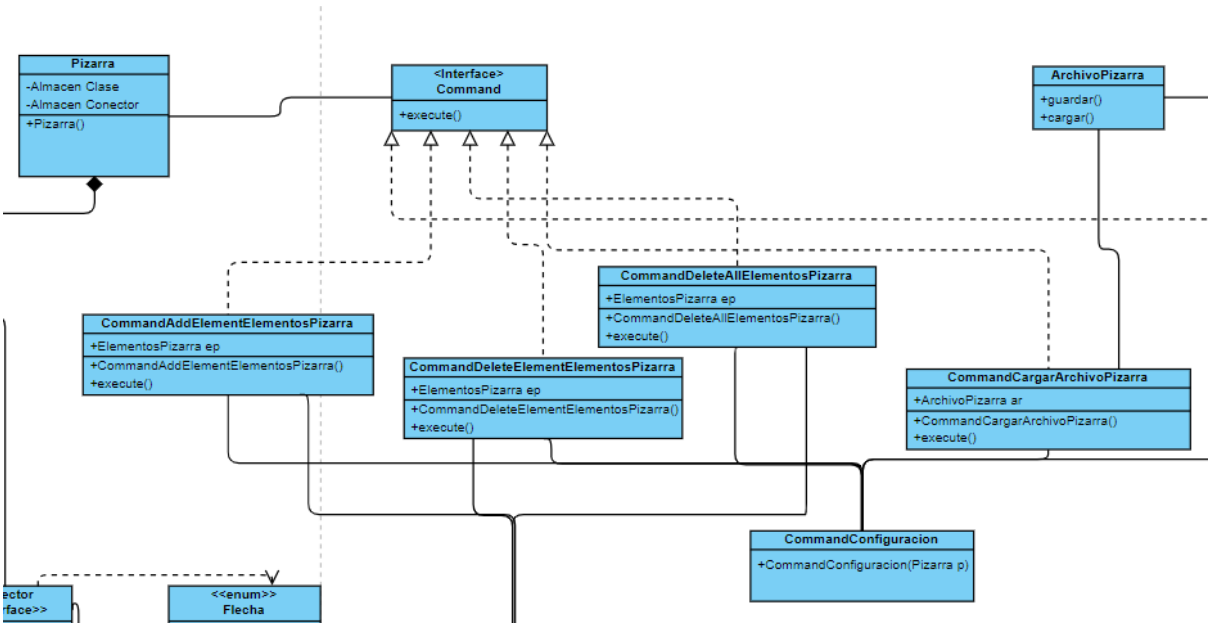
Diagrama de casos:

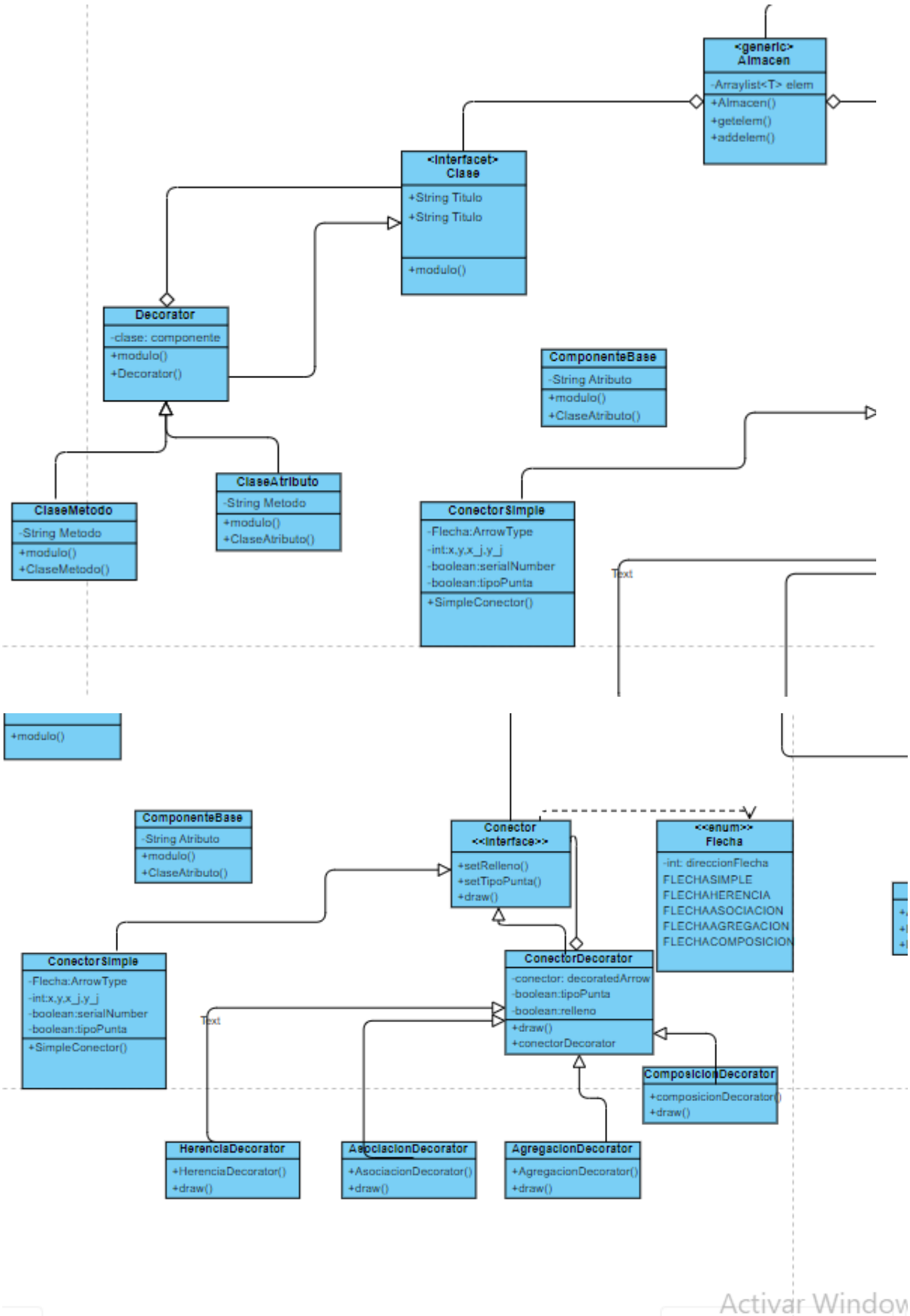


El siguiente diagrama es una representación gráfica que describe cómo interactúan los usuarios con nuestro sistema de pizarra interactiva.

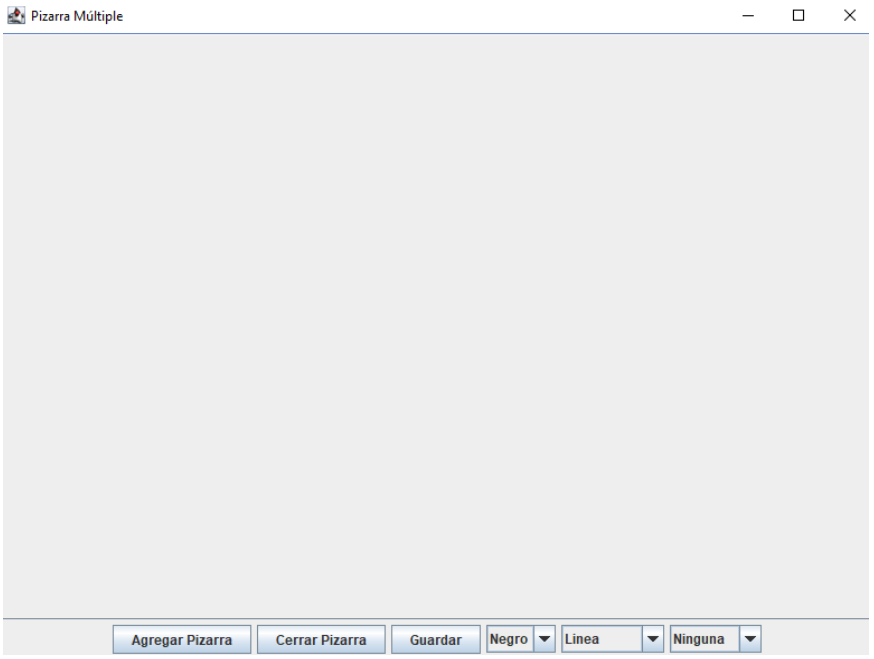
Diagrama UML (lógica):

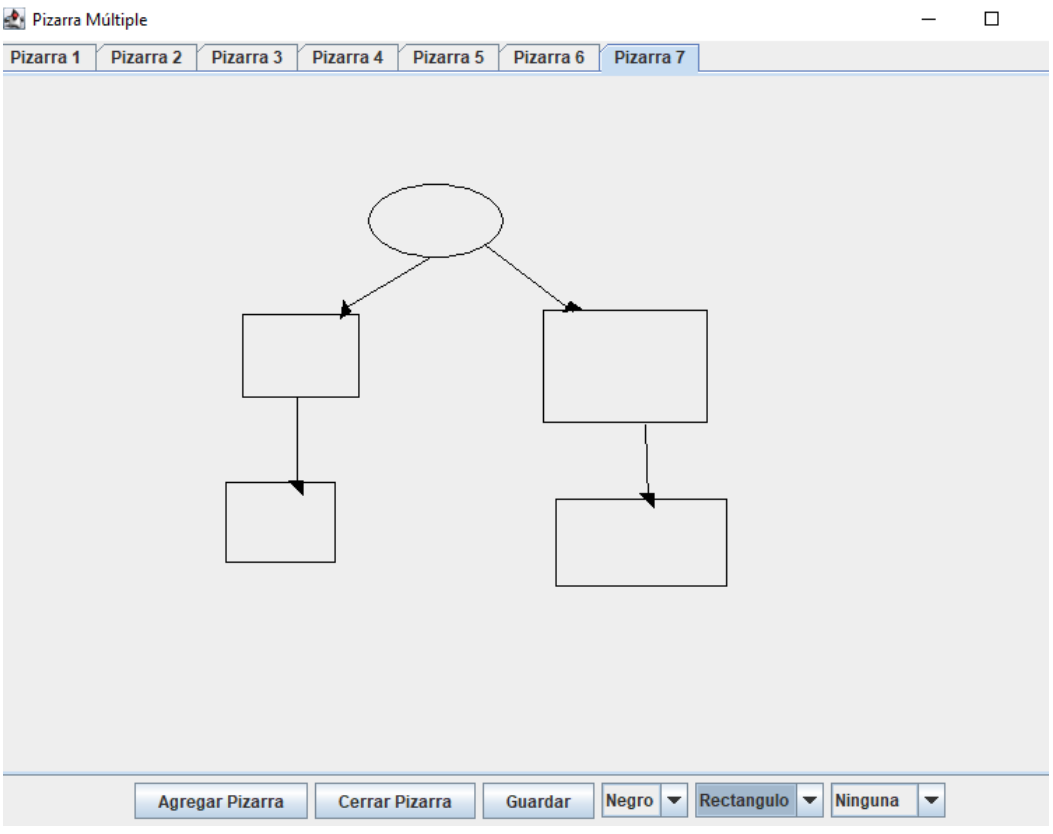


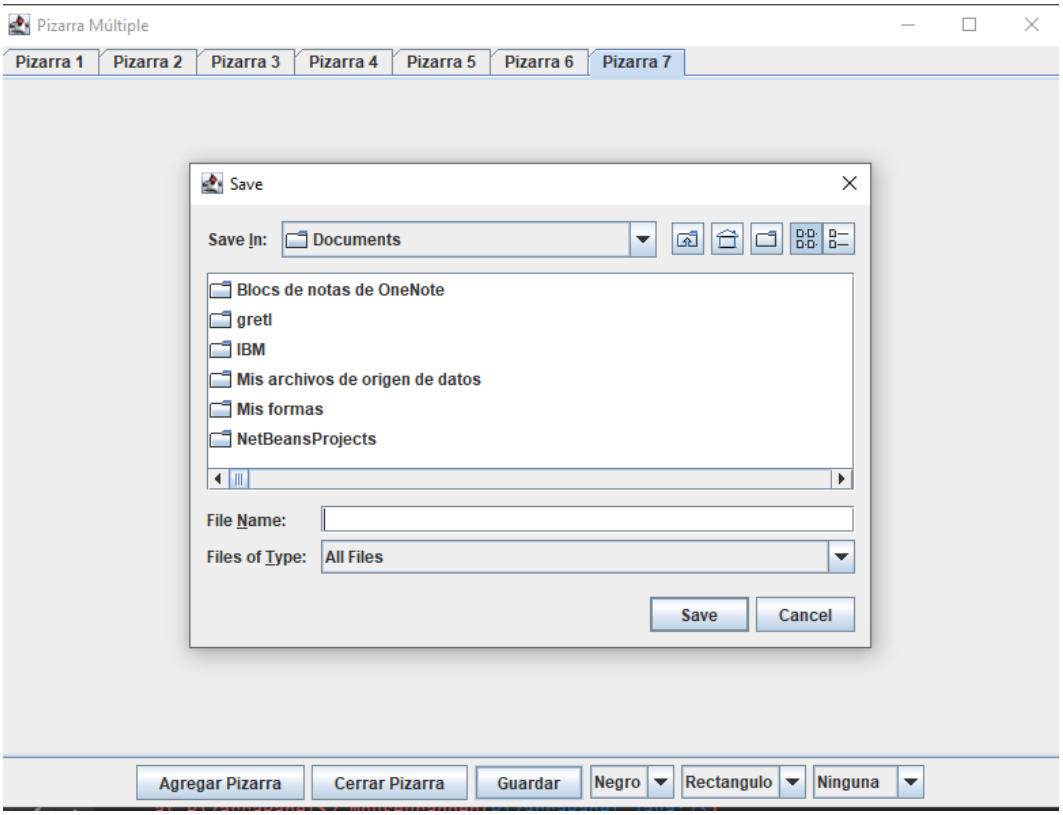




Captura de pantalla de la interfaz







PATRONES USADOS:

Decorator

En el desarrollo de este proyecto, hemos elegido utilizar el patrón Decorator en 2 instancias, como lo fue para implementar diversos tipos de flechas que sirven como conectores en el diagrama UML, como los de tipo simple, herencia, asociación, agregación y composición. Además, hemos aplicado este patrón en la creación de clases con diversas combinaciones de atributos y métodos.

Esta elección se fundamenta en la flexibilidad que el patrón Decorator proporciona para las funcionalidades de los objetos de manera dinámica. Su ventaja radica en la capacidad para incorporar nuevas funciones sin necesidad de modificar el código existente. Esto resulta particularmente útil dado que cada tipo de flecha posee características específicas, como diferentes tipos de puntas. Al utilizar decoradores especializados para cada tipo de punta, logramos adaptarnos fácilmente a las variaciones sin afectar la estructura principal del código.

El enfoque de utilizar decoradores no solo se limita a las flechas, sino que también se extiende a las clases implementadas, con sus respectivos atributos y métodos. Esto favorece la separación de responsabilidades, ya que cada decorador se encarga de una tarea específica, cumpliendo con el Principio de Responsabilidad Única. De esta manera, la estructura del código se mantiene eficiente y modular, facilitando la comprensión y mantenimiento a lo largo del desarrollo del proyecto.

COMMAND

En la creación de los botones en nuestro código, decidimos usar el patrón Command para hacer que la estructura sea más sólida y flexible. Esta elección se basa en razones que se relacionan directamente con las características y necesidades específicas de nuestro proyecto.

Principalmente, el patrón command tiene la utilidad de simplificar funcionalidades como lo son el guardar la pizarra y sus cualidades actuales utilizando la serialización, el cargar la pizarra con un nombre que se declara, el poder crear diferentes tipos de objetos ¨clase¨ y poder borrar todos los elementos de la pizarra. Todo esto con solo llamar a sus respectivos métodos a forma de botón, facilitando mucho su lectura e implementación de cada acción a largo plazo

Decisiones a tomar durante el proyecto

- La primera decisión que enfrentamos fue decidir la estructura y disposición de los elementos en la interfaz, como la ubicación de los botones, menús y pestañas.
- En cuanto al manejo de Eventos del Mouse se buscó definir cómo se gestionan los eventos del mouse para permitir el dibujo, arrastre y selección de elementos para ello se decidió la implementación de escuchadores de eventos del mouse.
- Por otro lado, debimos determinar la forma en que se gestionarán las pizarras múltiples, ya sea a través de pestañas en la interfaz o mediante un menú desplegable.
- Además decidimos cómo manejar los posibles errores al ejecutar el programa como la carga de un archivo incorrecto o una interrupción inesperada., de esta manera obtuvimos una aplicación más robusta, sumado a esto optamos por proporcionar una documentación detallada explicando cómo utilizar las diversas funciones de la aplicación, pues nos enfrentamos a una aplicación compleja

Problemas encontrado y autocrítica

Al desarrollar este proyecto, el mayor problema fue mantener el código organizado. El proyecto era complicado, y podría volverse confuso fácilmente. Tuvimos que revisar regularmente cómo estaba organizado el código para asegurarnos de entenderlo y poder hacer actualizaciones o correcciones fácilmente en el futuro.

También tuvimos problemas para coordinarnos como equipo, especialmente cuando teníamos que resolver conflictos en el código. A veces, los horarios de todos no coincidían, lo que retrasaba nuestro progreso.

Además, la parte de crear entidades y conectores UML resultó un poco complicada. Tuvimos que pensar en cómo diseñar las cosas de manera que fuera fácil agregar nuevas partes cuando lo necesitáramos.

Otra complicación surgió cuando uno de los miembros del equipo, Bianca Seminario, que había sido parte de un programa de intercambio, regresó a su país de origen. Esta situación generó obstáculos significativos en el avance del proyecto, ya que el miembro no pudo conectarse ni contribuir de manera activa durante un periodo de tiempo considerable debido a las limitaciones asociadas a su planificación con los vuelos a su país de origen y ajustes a su nueva ubicación. La falta de disponibilidad temporal del integrante mencionado y de comunicación previa al otro integrante sobre esta situación afectó de gran manera la dinámica del equipo y requirió que solo uno de los integrantes pudiera avanzar por un gran periodo del proyecto, como resultado, el resultado final fue bastante pobre y no esperado. La experiencia resalta la importancia de la planificación y comunicación para situaciones excepcionales para así poder adaptarse a los tiempos y avances requeridos. En resumen, trabajar bien juntos como equipo fueron los mayores desafíos.

También otro problema encontrado fue hacer el UML, ya que al ser tan grande, hay momentos donde se corrompía el archivo web o no se guardaba el progreso, por lo que se tuvo que hacer muchas veces el diagrama.

Conclusiones

El proyecto ofrece una valiosa oportunidad de aprendizaje al permitir la aplicación de conceptos avanzados de programación en Java, como el manejo de eventos y la interfaz gráfica. La resolución de problemas y tras la finalización del proyecto, es importante reflexionar sobre mejoras futuras, como mejoras en la interfaz de usuario, la inclusión de más formas UML o probablemente la implementación de funciones de zoom.

En el contexto de un desarrollo en equipo, la comunicación efectiva y el feedback entre los miembros son fundamentales para el éxito del proyecto, por otro lado nos dimos cuenta de la importancia de documentar el código de manera exhaustiva, realizando pruebas periódicas para asegurar el correcto funcionamiento de la aplicación.