Proyecto Semestral

(Tema 3: Tienda de Mascotas)

Desarrollo Orientado A Objeto

Integrantes:

Khristian Daniel Alexander Villalobos Alfaro

Máximo Ignacio Beltrán Aranzáez

Patrones de diseño utilizados y sus clases

1. **Command / EventBus**: El propósito de este patrón en nuestro contexto es permitir la comunicación entre la parte lógica y la parte visual de nuestro código, es una mezcla de ambos ya que permite tanto pedir como notificar cambio entre partes del código, como como comprar mascotas, notificar sobre un nuevo cliente, emplear gente, entre muchas otras funciones relacionadas, consiste en un "Bus" principal al que los objetos que implementen ciertas interfaces pueden suscribirse para enviar peticiones, recibir peticiones, o ambas, las clases que utiliza este patrón son:
    a. **EventHandler**: Esta clase es el "bus" de *Eventos*, es la columna vertebral del código, esta se encarga de recibir eventos de *Publicadores* y transmitirlos a una lista de *Suscriptores* que esta guarda.
    b. **Evento**: Esta clase abstracta guarda la información sobre cada evento o comando que pueden enviar los *Publicadores*, por cada evento posible existe una subclase de *Evento* que es enviada a los *Suscriptores* que escuchen.
    c. **Publicador**: Solo un publicador puede enviar eventos al "bus", o tener subclases que realicen esta función, esto no es forzado en el código, pero hacer cumplir este requerimiento permite tener en cuenta fácilmente quienes son las clases que pueden modificar el estado externo.
    d. **Suscriptor**: Un suscriptor es capaz de recibir eventos de uno o más *EventHandler* y modificar su estado en respuesta a estos.
    e. **TipoEvento**: Es una enumeración que guarda todos los tipos de eventos posibles, permite usar switch statements en los *Suscriptores.*

f. **DestinoEvento**: Es una enumeración que guarda un destino posible al que puede ir un evento, los *Suscriptores* tienen que decir cuales tipos de *DestinoEvento* admiten y cuáles no.

g. **Controlador**: Un controlador es una clase que envía y recibe eventos, pero que no guarda información importante por sí mismo, es un intermediario entre partes del sistema, por lo que implementa tanto *Publicador* como *Suscriptor*.

2. **Singleton**: Este singleton se usa para guardar imágenes cargadas, y que los objetos puedan pedir JPanel de cualquier dimensión con estas imágenes, esto es útil ya que las mismas imágenes se usan en varias partes del código, y esto permite extraer código repetitivo y reducir las operaciones IO, lo que ayuda a minimizar errores y hacer el código más legible:

    a. **ImageLoader:** Esta clase carga imágenes en un HashMap estático, y permite a otras clases pedir versiones escaladas de estas, y opcionalmente ponerlas en un componente Swing para ahorrar código.

Decisiones tomadas por el grupo

En la parte lógica del proyecto, la decisión mas importante que tomamos fue crear los controladores para no entregar objetos a los paneles, en la parte de cambios al enunciado del proyecto, nuestra primera decisión fue no crear una interfaz de cliente, sino que crear clientes de forma aleatoria, para que el usuario los atienda, de esta decisión naturalmente surgió otra duda, como hacer que aparezcan, nos pareció que poner una tasa fija de aparición seria aburrido, así que implementamos un sistema de calificaciones, el cual hace que mientras más calificación tiene la tienda, más gente viene a comprar, esta calificación depende de la atención dada a cada cliente, mientras mejor mantenida la mascota entregada, mejor calificación dará el cliente, esto hace nuestra tienda mas interactiva.

En la parte gráfica consistentemente ahorramos recursos re usando paneles después de creados, por ejemplo todas las ventanas principales son reutilizadas después de creadas, lo que ayuda a reducir el uso de memoria.

Desafíos enfrentados y autoevaluación

El consenso de nuestro grupo es que el proyecto fue un éxito, y que el trabajo fue repartido equitativamente, pese a esto encontramos ciertos desafíos menores a la hora de trabajar, tales como:

1. Las características modernas de Java, como los records y los streams, los que, aunque convenientes si uno sabe que está haciendo, pueden llegar a verse extraños al encontrarlos por primera vez.
2. La comunicación fue un problema para nuestro grupo, ya que el trabajo, aunque bien repartido, fue la mayor parte del tiempo de forma individual, pese a esto sentimos que no nos afectó tanto, ya que el código entregado es de buena calidad y nuestras ideas sobre el producto final eran similares.
3. La programación de interfaces gráficas de Java puede ser confusa e inconsistente, existen algunos lugares donde podríamos haber tomado mejores decisiones, o donde tuvimos que descartar algunas ideas que teníamos.