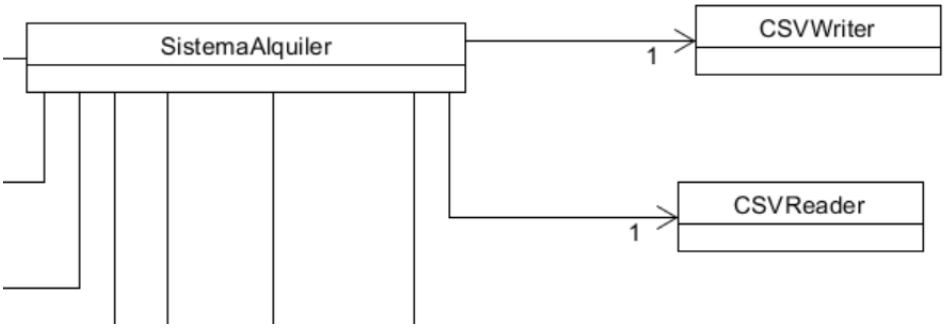
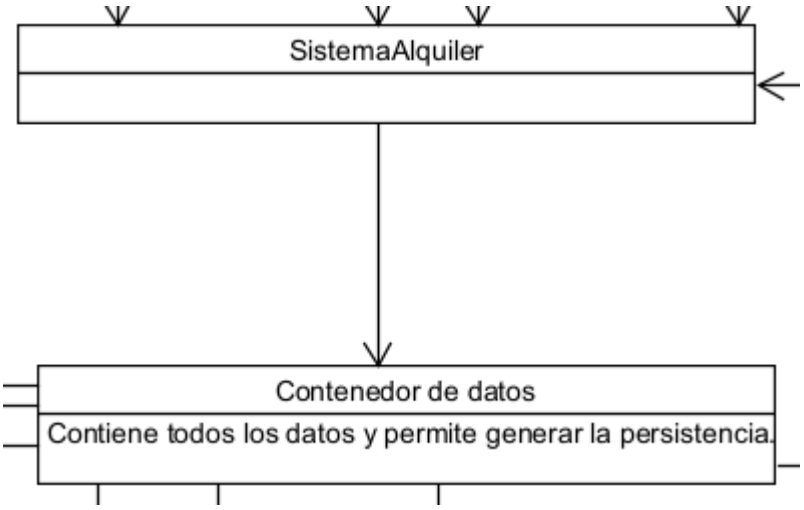
En este documento se va a reflexionar sobre el proceso de realización de los tres proyectos, enfocándonos específicamente en el proceso de diseño. Se profundizará sobre aquellos aspectos que salieron bien y mal, las decisiones acertadas y problemáticas, y los problemas que surgieron, indagando sobre estos el tipo de problema y su origen.

Reflexión diseño lógica de negocios

Implementación persistencias

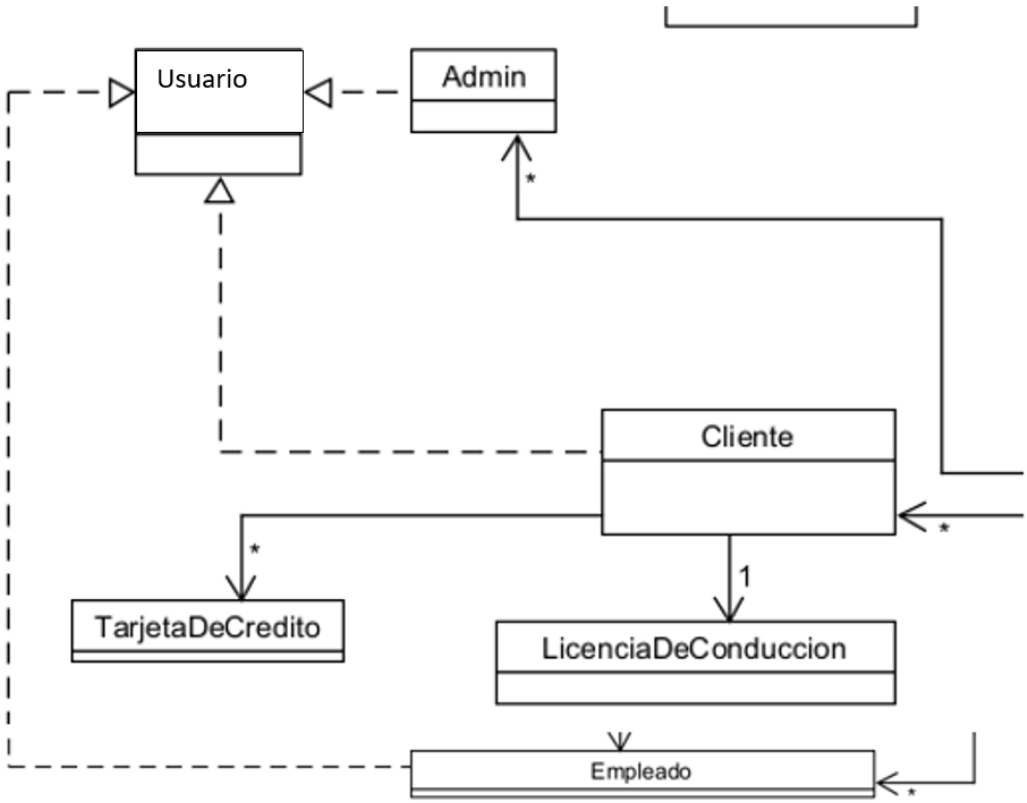


Hubiera sido útil implementar las clases CSVWriter y CSVReader siguiendo el patrón de diseño “Strategy”, ya que este se usa cuando es necesario definir una familia de algoritmos, encapsular cada uno de ellos y hacer que sean intercambiables. Además, esto hubiese facilitado un mayor desacople entre la carga de datos, puesto que en esta implementación si hay algún error durante la carga de algún archivo esto puede tener efecto de borde sobre la carga del resto.



No obstante, la implementación actual fue mejor que el uso de serializable (proyecto 1 y 2) ya que los CSV son más fáciles de leer e interpretar al ojo humano, que los archivos planos generados por serializable.

Implementación usuarios



Consideramos que la decisión de que las clases hereden de la clase Usuario fue una buena implementación para aprovechar la herencia y favorece la reutilización del código común a todos los usuarios. No obstante, hubiera sido una buena idea segregar más las interfaces, con el fin de tener interfaces específicas para "admin general" y "admin local", y a su vez para "empleado mantenimiento" y "empleado caja". Por ende, aplicar el Principio de Segregación de Interfaces hubiera sido mejor ya que en este una clase no debe está obligada a implementar interfases que no utiliza, lo cual permite tener una mayor claridad, especificidad, flexibilidad, escalabilidad, evita la sobre carga y a su vez permite un mantenimiento efectivo. Es por esto, que, aunque la herencia de la clase Usuario brinda una base común a todas las clases de usuarios, implementar la segregación de interfaces hubiera sido más beneficiosa al aportar las características mencionadas anteriormente.

Programa Clientes

La implementación de las clases asociadas a este programa resulto redundante, principalmente a causa de la implementación del Singleton GestorDatos. Una mejor forma de abordar este problema hubiera sido hace desde un inicio una clase de este tipo de patron para el proyecto en general, de manera que se garantice la seguridad de los datos y evitar duplicaciones. Igualmente, a futuro la implementación se hubiera podido ver beneficiada de tener mejor conocimiento de como sobrescribir métodos y manejar la herencia entre clases.

Implementación Componentes

Algunos componentes se implementaron de manera diferente, es decir, en algunos lados se utilizan componentes base como JButton y JText envez de componentes customizados, lo que significa que el programa se ve diferente dependiendo en la pagina que este el usuario. Adicionalmente en algunas páginas similares se utilizaron diferentes Layouts, lo que adiciona a este problema. Sin embargo, este es un problema puramente estético, lo que significa que el programa mantiene su funcionalidad, aunque no se vea 100% uniforme. Es algo que se puede mejorar, y hasta se pudo haber logrado si se hubiese coordinado mejor desde el comienzo, pero esto no se hizo también por falta de familiaridad con swing y los métodos que típicamente se utilizan para implementar interfaces utilizándolo.

Navegador

Se creo una clase llamada navegador, esta se encarga del flujo de navegación. La estructura general en ambas aplicaciones es, tener un Frame base que contenga paginas diferentes, y luego los componentes se ubican dentro de estas páginas. El navegador entonces es el componente base de la ventana del programa, y se encarga de agregar nuevas páginas, y de devolverse a la página anterior. Adicionalmente es el que contiene el mecanismo para mostrarle mensajes al usuario, por medio de algo parecido a un Toast en Android. La pagina base contiene una instancia de SistemaAlquiler y el navegador, y estos se pasan como parámetros a todas las demás paginas para que puedan devolverse a la página anterior, modificar el sistema de reservas, y mostrar un mensaje al usuario de ser necesario. Esta implementación es bastante sencilla y funcional, por lo que su diseño esta bien por ahora. En cuanto a cosas que se le pueden mejorar, tal vez el diseño del mensaje que se muestra a los usuarios podría ser mejor, y también no existe un mecanismo para prevenir que una página se devuelva todas las veces que quiera, pero por lo general si se implementa con cuidado y responsabilidad está bien.

Implementación PDFs

Para la generación de PDFs se utilizo la librería iText-Core con su extensión iText-pdfHtml. Esto permite crear un pdf con HTML como elemento de organización, en 5 lineas de código. Lo mas involucrado en este proceso es crear el HTML que se mostrara en el pdf, el cual lo programamos tal que únicamente tenga que recibir el id de la reserva en cuestión, y el cuerpo del documento que contiene la información de la reserva, el cliente, y el vehículo a arrendar. De esta manera el diseño del documento ya esta preestablecido, solo es cuestión de obtener los datos. En cuanto a funcionamiento, esta bien, tal vez se puede invertir mas tiempo en el diseño del documento, ya que actualmente esta relativamente simple.

Implementación Pasarelas

En cuanto a la implementación de las pasarelas, se creo un archivo de configuración llamado config.txt en la carpeta ./src/pagos. Este archivo almacena el nombre de cada clase completo, y el nombre de la pasarela. Se creo luego una clase abstracta llamada PasarelaPagos que establece las funciones que deberían implementar las pasarelas de pago, y adicionalmente tiene funciones estáticas que se pueden utilizar para hacer la llamada a la clase de la pasarela respectiva. Las clases de pasarelas respectivas están ubicadas en la carpeta ./src/pagos, y cada una se encarga de hacer la llamada a su api respectiva, y de imprimir los resultados a su archivo de historial de transacciones respectivo. Las funciones de cada pasarela reciben todos sus parámetros utilizando

VarArgs, esto permite que cada uno reciba una cantidad diferente de parámetros, la única restricción es que tienen que ser de tipo String, sin embargo, esto permite mucha flexibilidad y facilidad en cuanto a implementación. Con ánimo de abstraer el funcionamiento lo más posible, únicamente existe una clase para la interfaz que representa todas las pasarelas; esta recibe el nombre de la clase de la pasarela, la pasarela, y una vez se tiene esto se crea una instancia de la pasarela especifica, y esta se utiliza para obtener una lista de parámetros especifica a esta pasarela. Luego se crean los componentes de la interfaz dinámicamente basado en los parámetros obtenidos, y una vez los ingresa el usuario, se obtienen las entradas y se pasan los parámetros al objeto de la pasarela, a partir del cual se puede llamar la función obj.realizarPago(), el cual en practica haría la llamada a la api de pagos, y escribe la información de la transacción a su archivo de historial. En cuanto a esta implementación, es bastante fácil agregar otra pasarela, ya que únicamente es necesario implementar su clase que extienda PasarelaPagos, y agregar su nombre de clase y nombre al archivo de configuración, no es necesario implementar su interfaz ya que esta esta abstraída y al hacer estas dos cosas se mostrara en la pagina de pagos. Sin embargo, tal vez se pudo haber creado una clase intermedia que se encargue de crear los componentes de la interfaz, para poder crear elementos que no solo reciban texto, pero que también puedan verificar si están bien, y informarle al usuario si es necesario realizar cambios. Además de esto, es una implementación bastante practica que facilita mucho la adición de nuevas pasarelas.

Manejo de Errores

En cuanto al manejo de errores, puede que este se haya realizado de mejor manera. Actualmente existe una clase que le muestra un mensaje al usuario al realizar una acción en la interfaz, como por ejemplo al iniciar sesión, dice “sesión iniciada exitosamente”, y también al ocurrir un error, como por ejemplo al iniciar sesión también, si la contraseña o el usuario es incorrecto, muestra el mensaje “el usuario o contraseña es incorrecta”. Sin embargo, al ocurrir un error se retorna una excepción, esta se atrapa en un try/catch, y acá se obtiene el mensaje de error y se le muestra al usuario. La mejor manera de hacer esto hubiera sido crear un error diferente para cada error que puede ocurrir, tener un sistema para mirar que error acaba de ocurrir, y mostrarle al usuario el mensaje correspondiente, sin embargo, este proceso es bastante involucrado y requiere mucho más tiempo.