**Documento de autoevaluación**

La tabla proporciona una descripción detallada de los puntos a evaluar y los puntajes que se consideran apropiados.

|  |  |  |
| --- | --- | --- |
| **Item.** | **Concepto** | **Pts.** |
| 1 | Funcionalidad: Implementación de la funcionalidad solicitada en Java | 0 |
| 2 | Diagrama conceptual de dominio | 2 |
| 3 | Diagrama de diseño | 2 |
| 4 | División lógica | 6 |
| 5 | Experto | 6 |
| 6 | Fachada y arquitectura | 3 |
| 7 | MVC | 6 |
| 8 | Observador y manejo de eventos | 6 |
| 9 | Manejo de excepciones | 3 |
| 10 | Polimorfismo | 5 |
| 11 | Autoevaluación | 1 |

1. Funcionalidad: Implementación de la funcionalidad solicitada en Java

Se realizaron todas las funcionalidades solicitadas de manera eficaz, las cuales fueron probadas exitosamente utilizando los datos de prueba almacenados en el sistema. Además, se realizaron ingresos de datos manuales a través de la aplicación para garantizar su correcto funcionamiento.

1. Diagrama conceptual de dominio

Este aspecto se abordó de manera colaborativa durante la clase, en conjunto con el docente, con el objetivo de establecer un punto de partida común para los trabajos de todos los equipos.

1. Diagrama de diseño

A partir de nuestro diagrama original, procedimos a escribir el código de manera progresiva. A medida que avanzábamos en la solución, identificamos mejoras que debían ser reflejadas en el diagrama. Estas mejoras condujeron a la creación de nueve versiones, en las cuales se eliminaron clases previamente implementadas y se añadieron otras necesarias para la incorporación de los nuevos conceptos que surgieron a lo largo del curso.

En la solución final, el diagrama de diseño fue dividido en dos subdiagramas. Uno de ellos representa la Fachada, su interacción con los servicios y las clases de dominio. El otro subdiagrama muestra la relación entre la Fachada y las vistas de la interfaz de usuario (IU), los controladores, las interfaces y el estereotipo Observer (en las clases correspondientes).

1. División lógica

La implementación de la división lógica en nuestro proyecto se realizó correctamente, siguiendo los principios de la división lógica de software. Se estableció una clara separación de responsabilidades y comunicación entre las diferentes capas del sistema. En particular, se definió una capa de presentación (IU) encargada de la interacción con los usuarios, una fachada que actúa como intermediario entre la capa de presentación y la capa del dominio, y finalmente, una capa del dominio donde se concentra la lógica de negocio y los componentes principales del sistema.

1. Experto

Durante el desarrollo de nuestro código, nos encontramos con varias dudas acerca de cómo aplicar el patrón experto, especialmente en el contexto del cálculo del monto a pagar por tránsito. Este cálculo involucraba datos de varias clases, lo que dificultaba determinar cuál era la clase experta adecuada para llevar a cabo esta tarea.

Concluimos que la clase "Tránsito" era la opción más adecuada para implementar el cálculo. Esta elección se basó en la aplicación del principio experto, el cual establece que una clase debe ser responsable de llevar a cabo tareas relacionadas a su propia información. En este caso, la clase "Tránsito" contiene los datos necesarios y tiene el conocimiento necesario para realizar el cálculo del monto a pagar. Por lo tanto, consideramos que el cálculo fue correctamente aplicado en la clase "Tránsito" siguiendo el patrón experto.

1. Fachada y Arquitectura

En nuestro proyecto, hemos implementado correctamente el patrón Fachada, el cual nos permite ocultar la lógica de negocio al usuario. Mediante la Fachada, se proporciona un punto de comunicación único entre la interfaz de usuario (UI) y la capa del dominio. Además, hemos aplicado el patrón Singleton en la implementación de la Fachada, asegurando que exista solo una instancia de esta clase en el sistema.

Además de la aplicación de patrones, hemos dividido físicamente el código en diferentes paquetes, lo cual nos ha permitido organizar el proyecto de manera efectiva. Esta división facilita la realización de cambios o modificaciones, ya que cada paquete contiene componentes relacionados y permite una comprensión clara de la estructura del código.

También hemos utilizado nombres de métodos descriptivos, lo que contribuye a mejorar la legibilidad y comprensión del código. Los nombres de métodos expresan claramente su función y propósito, lo que facilita el mantenimiento y la colaboración en el desarrollo del proyecto.

1. MVC

En nuestro proyecto, hemos incorporado correctamente el patrón Modelo-Vista-Controlador (MVC). Este patrón ha permitido asignar a cada clase las funciones correspondientes que antes se encontraban presentes en las interfaces de usuario (UI).

Al atribuir adecuadamente las funciones a cada una de las clases en nuestro proyecto, hemos logrado una separación clara de responsabilidades. El modelo se encarga de manejar los datos y la lógica de negocio, la vista se encarga de la presentación visual y la interacción con el usuario, y el controlador coordina y gestiona la comunicación entre ambos.

Esta implementación del patrón MVC nos ha permitido mejorar la modularidad, reutilización y mantenibilidad del código, al separar las diferentes partes del sistema y asignarles funciones específicas. Asimismo, al remover la lógica de negocio de las interfaces de usuario, hemos logrado una mayor coherencia y flexibilidad en el diseño de la aplicación.

1. Observador y manejo de eventos

La aplicación de este patrón en el proyecto se ajustó a las expectativas establecidas. No obstante, en ciertos momentos nos encontramos con dificultades para identificar la línea de código precisa en la cual debíamos disparar el evento. Sin embargo, encontramos útil la nomenclatura utilizada para identificar los eventos, ya que nos proporcionó una mayor comprensión del lugar exacto donde se debía realizar la llamada. Algunos ejemplos de los nombres que utilizamos son "RECARGA\_SOLICITADA" y "RECARGA\_APROBADA".

Esta nomenclatura descriptiva facilitó la comprensión del código y permitió ubicar de manera efectiva los puntos en los que se debían disparar los eventos. Esto garantizó que los observadores adecuados fueran notificados y tomaran las acciones correspondientes en respuesta a los eventos.

1. Manejo de Excepciones

La incorporación de excepciones en el proyecto se realizó de manera adecuada, siguiendo los cursos alternativos solicitados en el enunciado del obligatorio. El uso del bloque "try-catch" y el uso de la declaración "throws" nos proporciona un manejo más efectivo para comunicarnos con el usuario en caso de que se produzca algún problema específico y evitando que el programa se detenga de forma abrupta.

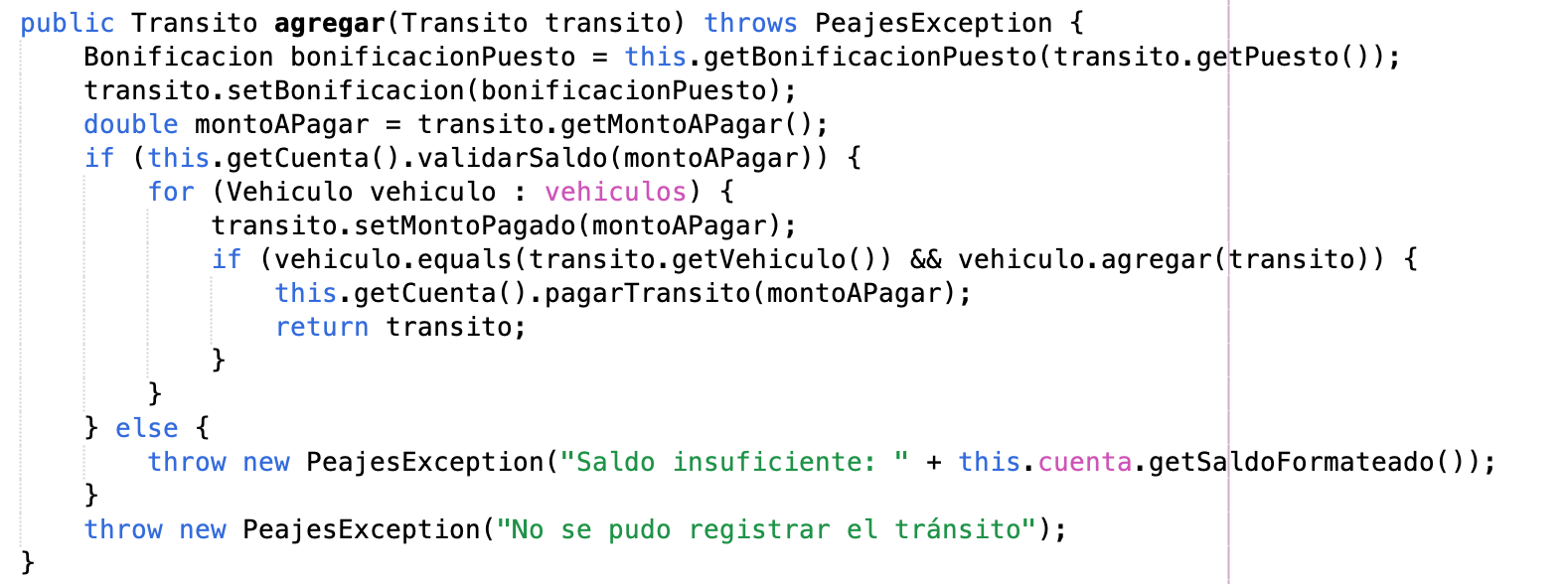
1. Polimorfismo

En el caso del login, aplicamos el principio de polimorfismo para adaptar la lógica de ingreso al sistema según las diferentes credenciales presentadas. Utilizamos un objeto que puede tener múltiples formas, lo que significa que puede comportarse de manera distinta en función de las credenciales proporcionadas. De esta manera, el código que se ejecutará será la implementación específica correspondiente a las credenciales presentadas.

Al aplicar el polimorfismo en el login, podemos aprovechar la flexibilidad y extensibilidad de la programación orientada a objetos. Esto nos permite adaptar el comportamiento del objeto de acuerdo con las necesidades específicas de cada caso, sin tener que modificar el código existente. En lugar de tener múltiples condicionales que verifiquen las credenciales y ejecuten diferentes bloques de código, utilizamos el polimorfismo para seleccionar automáticamente la implementación correcta en función de las credenciales presentadas.

1. Autoevaluación

En este punto repasamos nuestro código y detectamos un único método (agregar) que quizá se podría mejorar de forma visual para que su lectura sea más clara ya que nos parece un poco extraño el tener un bucle entre dos condicionales y el código sea largo. El mismo es el siguiente (../modelo/UsuarioPorpietario):



Por otro lado, no pudimos incorporar de forma correcta los iconos a las UI para mejorar la UX del sistema.

Fuera de estos dos puntos mencionados, consideramos que nuestro trabajo cumple de manera satisfactoria los requisitos propuestos en la letra del mismo.