Programación con Objetos II

Trabajo Final: Sistema de Estacionamiento Medido

Integrantes:

- Ariel Murua (arielm.murua@gmail.com)
- Lautaro Coria (lautarocoria97@gmail.com)
- Matías N. Fuentes (matiasnfuentes@gmail.com)

Profesor: Diego Torres

Universidad: Universidad Nacional de Quilmes

Año: 2020

Cuatrimestre: 2º

Patrones de diseño utilizados:

1. Monitoreo de estacionamientos:

Uno de los requerimientos del trabajo practico era que "el municipio desea establecer un esquema de suscripción/desuscripción, muy genérico, que permita que distintas entidades o sistemas sean alertados ante distintos eventos del sistema."

Para poder implementar este requerimiento, aprovechamos la utilización del patrón **observer**, que nos permitía cumplir con el mismo de forma adecuada. Los roles y las clases utilizadas fueron las siguientes:

Subject: **AlarmaDeEstacionamiento** , quien provee una interfaz para añadir o quitar observadores, es quien los conoce.

Observer: **Observador** (interfaz), indica los métodos que tienen que implementar los ConcreteObservers.

ConcreteObserver: No hay ningún observador concreto en nuestra implementación. Estos serán aquellos que implementen la interfaz observer (es decir quienes se suscriban a los avisos).

ConcreteSubject: En este caso son dos: **AdministracionDeEstacionamientos** y **AdministracionDeCelulares**. Indican cuando uno de sus estacionamientos inicia o termina, y cuando hay una recarga de crédito en uno de sus celulares , respectivamente.

Ademas, para evitar valores nulos en la solución, decidimos implementar el patrón **null object**, y la clase que implementa este rol es **EstacionamientoNulo.**

2. Modo manual-automático:

Otro de los requerimientos del trabajo práctico era que la app usuario tenga un modo automático, que iniciaba y finalizaba estacionamientos automáticamente, cuando detectaba cambios en los desplazamientos, y un modo manual, en el cual se inician los estacionamientos manualmente, pero se reciben notificaciones si se detecta un posible inicio o finalización de estacionamiento.

Para cumplir con este pedido, implementamos un patrón **Strategy**, donde la estrategia que adopta cada modo ante un posible inicio o fin de un estacionamiento queda encapsulada en una clase especifica (una estrategia específica). Los roles y las clases utilizadas fueron las siguientes:

Strategy: La clase que implementa este rol es la clase **Modo**, que define la interface común a todos los algoritmos.

ConcreteStrategy: Las clases que cumplen este rol son **ModoAutomatico** y **ModoManual**, quienes implementan los algoritmos usando la interfaz del strategy.

Context: La clase que cumple este rol es **AppUsuario**, que se configura con una de las estrategias concretas.

3. Detección de desplazamiento:

Otro de los puntos del trabajo era que los usuarios puedan activar la detección de

desplazamiento para que se pueda manejar automáticamente (o no) los cambios de desplazamiento en automóvil o a pie.

Para llevar a cabo este requerimiento, implementamos un patrón **State**, que tiene las clases Caminando y Manejando, y ambas implementan una interfaz para que determinen que deben hacer en cada caso, mientras estén manejando o caminando. Los roles y las clases utilizadas fueron las siguientes:

State: La interfaz que implementa este rol es **EstadoDelUsuario**.

StateConcreto: Las clases que implementan este rol son **Manejando** y **Caminando**.