

TRABAJO PRÁCTICO FINAL

Programación Orientada a Objetos II

Integrantes:

- Lucas Paolo (Email: lucaspaulo002@gmail.com).
- Fernando Romero (Email: agustinfernando2000@hotmail.com).
- Mauricio Velazquez (Email: mauriciovelazquez211@gmail.com).

Decisiones de diseño:

- Al momento de realizar el gps este lo podríamos haber implementado como si fuera una interfaz, pero ya que, solo nos devolvía la zona en la cual se encontraba hicimos que simplemente la AppUsuario posea esta zona.
- Incorporamos la clase Reloj, que devuelve la hora actual para poder verificar los horarios de los estacionamientos, tanto para iniciar como para finalizar en los tests con Mockito.
- En el patrón Observer se notifica dependiendo del evento que suceda a todas las entidades suscriptas.
- CompraPuntual y RecargaCelular ambas son Data Class que proporcionan abstracción.
- Cuando se activa el modo automático la asistencia también se activa.

Patrones utilizados:

Patrón Observer:

Aplicamos este patrón para que los interesados en recibir notificaciones de inicio, fin de estacionamiento o recarga de saldo puedan suscribirse/desuscribirse del sistema cuando quieran.

Los roles entonces son:

- La clase **SEM** como el **Observable Concreto**.
- La clase **AppUsuario** con el rol de **Contexto**.
- La interfaz **INotificador** con el rol de **Observable**.
- La interfaz **ISuscriptor** con el rol de **Observer**.

Patrón State:

Aplicamos este patrón porque el ModoApp envía mensajes automáticos para iniciar y finalizar estacionamientos que dependen de si el usuario está estacionado o no.

Los roles entonces son:

- La clase **AppUsuario** con el rol de **Contexto**.
- La interfaz **EstadoEstacionamiento** con el rol de **State**.
- Las clases **EstacionamientoIniciado**, y **EstacionamientoSinIniciar** con el rol de **Estados Concretos** que implementan la interfaz EstadoEstacionamiento.

Patrón Strategy:

Aplicamos este patrón porque la AppUsuario contiene dos formas de iniciar y finalizar un estacionamiento (Manual/Automático) y el usuario de la misma puede elegir cual quiere.

Los roles entonces son:

- La **AppUsuario** con el rol de **Contexto**.
- La interfaz **ModoApp** con el rol de **Strategy**.
- Las clases **ModoAutomático** y **ModoManual** con el rol de **Estrategias Concretas** que implementan la interfaz ModoApp.

Como cada estrategia es independiente una de otra, se puede agregar en cualquier momento otro criterio.

Patrón Strategy:

Aplicamos este patrón porque la AppUsuario además del Modo contiene dos formas de asistir al usuario que puede estar (Activada/Desactivada) y el usuario de la misma puede elegir cual quiere.

Los roles entonces son:

- La **AppUsuario** con el rol de **Contexto**.
- La interfaz **Asistencia** con el rol de **Strategy**.
- Las clases **AsistenciaActivada** y **AsistenciaDesactivada** con el rol de **Estrategias Concretas** que implementan la interfaz Asistencia.

Como cada estrategia es independiente una de otra, se puede agregar en cualquier momento otro criterio.