Trabajo Práctico Final Objetos II - Grupo 11

Integrantes:

- Tomás Cowes (tomascowes97@gmail.com)
- Nicolás Nieto Cowes (nicolasnietocowes@gmail.com)
- Luciano Santoleri (luchosantoleri@gmail.com)

Decisiones para el diseño:

Comenzamos identificando qué clases iban a ser de utilidad para nuestro diseño en base a lo mencionado en el enunciado. Determinamos que íbamos a precisar de las siguientes clases:

- *SEM:* el encargado de registrar los estacionamientos generados en una colección, junto con las compras que se hacen en los distintos puntos de venta, las zonas de estacionamiento y las infracciones. Además cuenta con la función de finalizar todos los estacionamientos por parte de un operador (controlando el fin de franja horaria estipulado por quien utilice el sistema).
- *AppCelular:* según lo indicado en el enunciado, puede tener dos modos: Manual o Automático. Estos son seleccionables por el usuario y algunas de las funciones dependen de en qué modo se encuentra en determinado momento. Está asociada a un sistema de manejo de Estados (Caminando o Manejando) que se basa en un sensor de movimiento propuesto por el propio trabajo. Este Estado también hace

que varíe el comportamiento de la app. Provee funciones básicas como la carga de crédito de estacionamiento y otras complejas (como iniciar o finalizar un estacionamiento) que dependen tanto del modo como del estado en el que se encuentra.

- *Estacionamiento:* una superclase abstracta que se divide en estacionamientos por compra puntual y estacionamientos iniciados vía app. Cada subclase responde a métodos en común que presenta la superclase abstracta de forma distinta (por ejemplo si está vigente o la hora de finalización del mismo).
- *Compra:* otra superclase abstracta que se divide en los dos tipos de compras que puede haber en el sistema: la recarga de un celular y la compra de horas puntuales para estacionar. Cada subclase tiene un comportamiento diferente. La compra de horas puntuales directamente genera un estacionamiento por compra puntual y la recarga de celular le envía crédito de estacionamiento a una app que tenga asociada ese número de teléfono.
- *Punto de Venta:* se encarga de registrar las compras y enviarlas al SEM
- *Zona de Estacionamiento:* representa una zona de estacionamientos que va a estar asociada a diversas clases del sistema.
- *Inspector:* se encarga de consultar con el SEM la vigencia de estacionamiento contra una patente específica. También puede emitir Infracciones que luego el SEM registra, todo en el caso de que la patente no tenga un estacionamiento activo como se mencionó previamente.
- *Infracción:* representa una infracción con los datos solicitados en el enunciado.

Detalles de implementación:

Determinamos que una precondición para el manejo de los estacionamientos iba a ser el control previo del horario de finalización de cada uno. Es por eso que se requieren diversos accesos al SEM para obtener el finDeFranjaHoraria determinada por este (decidimos que esto correspondía a un método específico, en el cual usamos el horario que se menciona en el enunciado, las 20:00hs, pero cuenta con la flexibilidad para cambiarlo si fuera necesario) y de esta forma poder hacer la comparación.

Realizamos una interpretación para traducir el valor en pesos a créditos de estacionamiento, de modo que estos créditos representen minutos y de esta manera se simplifique el control con las horas transcurridas dependientes de un crédito específico (asociado a una app). La relación se puede representar como:

1 peso -> **1.5** crédito de estacionamiento
40 pesos -> **60** créditos de estacionamiento*

*el equivalente al valor de la hora de estacionamiento.

Patrones de diseño:

Dimos uso de los siguientes patrones de diseño:

- En la clase Estacionamiento utilizamos Abstract Methods para que las subclases definan el comportamiento esperado de cada una.
- Lo mismo sucede en la clase Compra y sus subclases, con template methods y jerarquía de clase.
- A su vez, la Clase PuntoDeVenta interactúa con la clase Compra y sus subclases para llevar a cabo el registro de las compras de estacionamiento a través del SEM.
- En la AppCelular se da un caso particular, como el Estado de la misma depende de estar observando constantemente qué mensajes “podría estar enviando” algún agente externo que tenga que ver con el sensor de movimiento o un sistema de GPS. Por lo que se podría decir que se presenta un patrón Observer simple en el cual la app cumple el rol de observador (nuevamente, faltaría la clase observable que se corresponde con el GPS). Esto no se encuentra implementado porque el enunciado aclara que no es necesario: “Usted no debe preocuparse por enviar estos mensajes a las app, asuma que llegan; sí debe encargarse de recibirlos y gestionar la app en base a ellos.”
- En el ModoApp se utiliza un patrón Strategy, en el cual la AppCelular cumple el rol de contexto y los modos ModoManual y ModoAutomatico son las estrategias concretas que heredan de la estrategia ModoApp.

- En EstadoApp utilizamos un State Pattern, nuevamente la AppCelular cumple el rol de contexto y el estado es la clase EstadoApp (abstracta), que se divide en las subclases Manejando y Caminando que representan los estados concretos (interactúan entre ellos ya que cada uno puede cambiar el estado actual de la app a su contraparte).
- En la interfaz propuesta como IMonitoreoEstacionamiento utilizamos un Observer Pattern, teniendo a la clase SEM que administra los estacionamientos como clase observable y cualquier otra clase que implemente la interfaz IMonitoreoEstacionamiento (y Observer) como observadora.