INFORME DEL TRABAJO PRÁCTICO INTEGRADOR

Integrantes:

- Stambul, Leonel Axel: mailto:leoquilmes88@gmail.com
- Montiel, Leonel Elias: leonel.montiel@outlook.com

Link al repositorio:

- leonelmontiel/TP-Integrador-POO2-2021 (github.com)

DESICIONES DE DISEÑO

Hemos diseñado nuestro sistema teniendo en cuenta los cinco principios recomendados por Robert C. Martin, para poder lograr un proyecto estable y eficaz.

A continuación, se detallarán estos principios y sus implementaciones en el desarrollo integral del sistema:

Single Responsibility Principle (SRP)

Su principio reza que "una clase debería tener solo una razón para cambiar". Por este motivo, en el *SEM*, separamos los comportamientos que cambian por razones diferentes. Por ejemplo, los que tienen que ver con los estacionamientos son definidos por su propia interfaz (*GestorEstacionamiento*), lo mismo sucede con las aplicaciones, infracciones y registros (*GestorAPP, GestorInfracciones, GestorRegistros*). De esta manera evitamos una clase *SEM* que englobe comportamientos completamente diferentes, en los cuales si se realizara algún cambio provocaría la modificación en otra responsabilidad acoplada.

Open/Closed Principle (OCP)

Respetamos este principio para no tener que añadir nueva lógica en distintos componentes del sistema cuando decidamos extenderlo. Por ejemplo, tenemos la posibilidad de agregar tantos tipos de *Estacionamiento* como quisiéramos, además de los dos que ya están implementados (*EstacionamientoAPP y EstacionamientoCompraPuntual*). Esto mismo sucede con los de tipo *RegistroDeCompra* y *EstadoAPP*.

Liskov Substitution Principle (LSP)

Intentamos que nuestro programa tuviera un código que se pueda reutilizar y una jerarquía de clases fácil de comprender. Por ende, toda subclase conoce e implementa, a su modo, los mismos mensajes que define su superclase. De esta manera podríamos realizar testeos de cualquier subclase sin importar a qué mensaje de la clase padre se llame, y esta responderá de manera esperada de acuerdo al propósito de la implementación.

Interface Segregation Principle (ISP)

Si bien en nuestro sistema solo hay una clase cliente (SistemaCentalImpl) que implementa más de una interfaz de a cuerdo con los roles que cumple en relación a las partes del mismo, también

se cuenta con otras que representan un rol de sus partes permitiendo que se reemplace sus implementaciones sin afectar el cumplimiento de las responsabilidades del sistema ni su funcionalidad. De este modo también si un cliente necesitara asumir un nuevo rol, podría implementar una interfaz ya existente o incorporar una que represente las funcionalidades requeridas.

Dependency Inversion Principle (DIP)

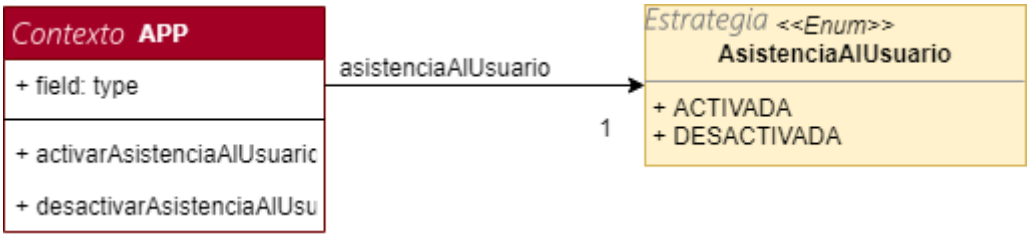
Esto sucede con todos los gestores, pero tomamos de ejemplo el *GestorAPP*. El gestorAppImpl es un módulo de bajo nivel que no interactúa directamente con el *SistemaCentralImpl* que es de alto nivel, sino que implementa una abstracción, es decir, la interfaz *GestorAPP*. Esta interfaz es la que sí interactúa con el módulo del sistema central, que, a su vez, implementa una abstracción para lograr el mismo efecto con aquellos módulos que dependan de esta. Entonces, sin importar el tipo de gestor de app que se le pase al sistema central, ni este ni sus instancias tendrán que cambiar.

PATRONES IMPLEMENTADOS

Strategy (2 veces en *AsistenciaAlUsuario* y en *ModoAPP*): Depende del usuario.

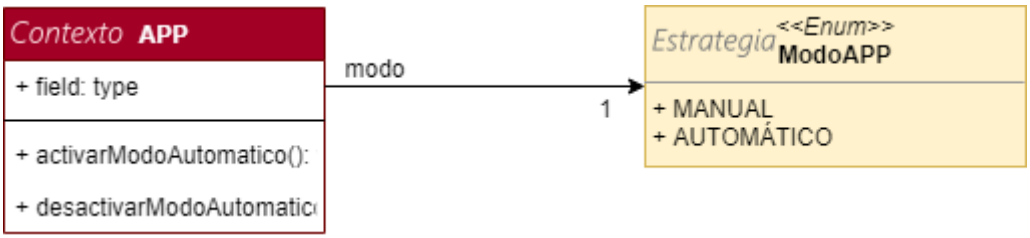
- AsistenciaAlUsuario:

  se delega la toma de decisión de alertar al usuario ante los eventos disparados por algún cambio de movimiento por parte del mismo, a estas instancias de enumerativo (*AsistenciaAlUsuario*) quienes determinan si se debe lanzar la notificación o no. El usuario puede decidir mediante el protocolo dado por la *APP* ("ACTIVADA" y "DESACTIVADA") la manera en la que se efectúe este comportamiento.



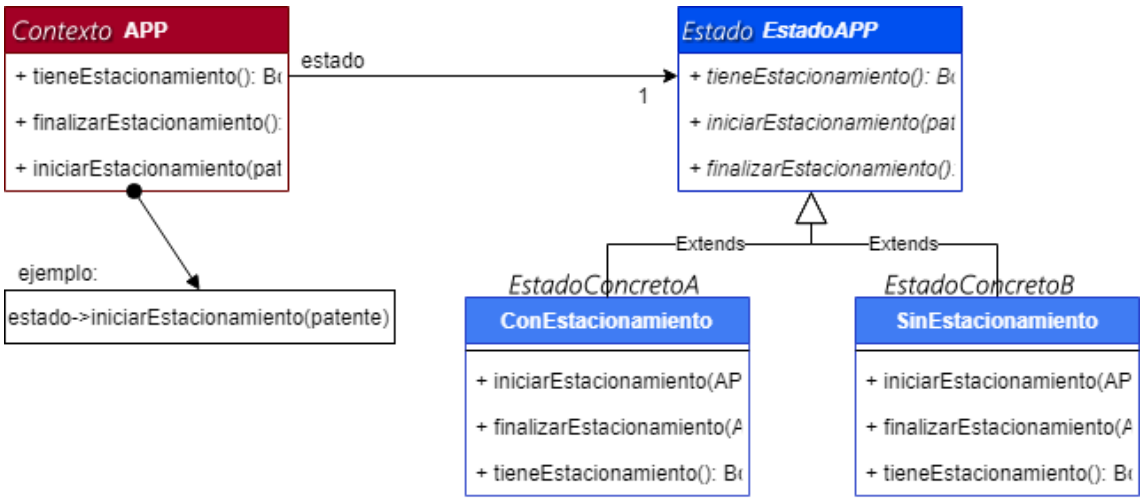
- ModoAPP:

  se delega la toma de decisión de iniciar o finalizar un estacionamiento ante los eventos disparados por algún cambio de movimiento por parte del usuario, para que este pueda decidir si realizarlo de manera manual o automático. El usuario puede cambiar en cualquier momento el modo mediante los protocolos dados por la *APP* ("MANUAL" Y "AUTOMÁTICO").



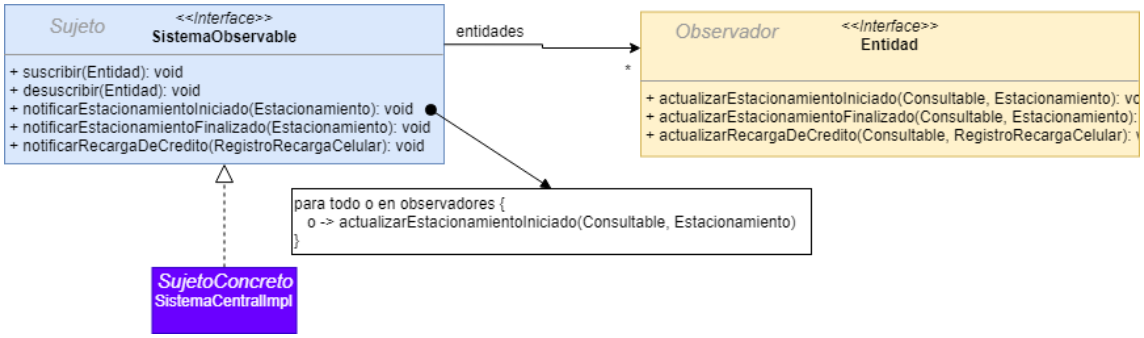
State (en *APP*: *ConEstacionamiento* y *SinEstacionamiento*):

Muta de manera automática. El módulo *APP* está dotado con la capacidad de controlar el estado en el que se encuentra y delega en este los comportamientos que deben llevarse a cabo al momento de iniciar/finalizar un estacionamiento.



Observer (*SistemaCentralImpl* suscribe a *Entidad*):

El sistema central puede suscribir y desuscribir entidades, para poder enviarles alertas por cambios en algún aspecto del mismo. Estas entidades que implementan la interfaz reciben el mensaje que forma parte del protocolo firmado y que les permite hacer uso de la información relacionada al cambio de estado del sistema central. Específicamente en este caso, reciben la actualización ante un estacionamiento iniciado, uno finalizado o una recarga de crédito, pudiendo implementar las acciones que le sean de interés a dichas entidades a partir de esta notificación.



Singleton:

Para los estados se aplicó el patrón *Singleton* de construcción que evita contar con múltiples instancias de estos por cada aplicación contenida en el sistema. Se tomo esta decisión teniendo en cuenta que las apps solo pueden encontrarse en uno de estos estados a la vez, y que pueden ser representados unívocamente por una instancia de estos.

DETALLES DE IMPLEMENTACIÓN

- En una primera versión, habíamos desarrollado una clase *SEM* que integraba responsabilidades de todo tipo, tanto para registrar aplicaciones, estacionamientos, compras, recargas, infracciones, como también almacenar estos componentes.

Lo negativo de esta implementación es que rompe con el principio de única responsabilidad, por eso hemos decidido realizar una modificación significativa para solucionar esta cuestión. La segregación en interfaces fue lo que resolvió esta violación al principio, hemos dividido las obligaciones en ocho interfaces que se comunican con el *SEM*, las cuales proveen protocolos específicos:

- *GestorAPP*: para recargar saldo, iniciar y finalizar estacionamientos;

- *GestorEstacionamiento*: para generar, registrar y consultar estacionamientos;

- *GestorInfracciones*: para consultar y dar de alta infracciones.

- *GestorRegistros*: para generar recargas y compras puntuales, y almacenarlos.

- *Administrable*: para configurar la hora de inicio, la de cierre, el precio por hora y finalizar todos los estacionamientos.

- *Consultable*: para consultar la hora de inicio, la hora de cierre, el precio por hora y el costo de un estacionamiento. Esto con la idea de agregar una capa de seguridad.

- *SistemaObservable*: para suscribir, desuscribir y notificar entidades.

- *SistemaCentral*: es la principal, con la que interactúan las demás interfaces, esta define los mensajes para generar, iniciar y finalizar estacionamientos. Es decir, define el comportamiento de aquel componente que represente la totalidad del sistema manejando las interacciones entre los distintos tipos de gestores.

La decisión de incluir estos componentes es para que cada aspecto del sistema central interactúe con las partes que les corresponde por separado, evitando así un *SEM* generalizado. Por ejemplo, el aspecto que se encarga de administrar el sistema, en vez de interactuar con la interfaz completa *SistemaCentral*, lo hace con la interfaz *Administrable*.

- Se aislaron los estacionamientos iniciados por *APP* de los de compra puntual, con la intención de no filtrarlos de una lista que almacene integralmente los dos tipos de estacionamientos. Así de esta forma se puede interactuar con ellos exclusivamente comunicándose con la clase que implemente *GestorAPP*.
- El diseño general con la segregación de interfaces implementada permite un sistema modularizado, donde no habría mayores complicaciones si fuera necesario agregar alguna funcionalidad en el sistema.
- Se agregó una interfaz *Pantalla* para emular la interacción entre la *APP* y el usuario, y el *PuntoDeVenta* y el usuario.

OBSERVACIONES

- Posiblemente la interfaz del *SistemaCentral* puede subdividirse en distintas interfaces.

- Hay comportamientos que algunas interfaces comparten, lo que generaría la creación de nuevas interfaces...