Informe Trabajo Integrador

En este informe se especificará las decisiones de diseño, detalles de implementación, patrones de diseños utilizados, etc.

Alumnos: Luciano Cardozo Casariego, Ramiro Gonzalez, Barbara Silva Mesquita

Conflictos con GIT

Durante el desarrollo del proyecto nos encontramos con varios conflictos al utilizar git.

El primero de ellos fue no poder incluir como colaborador a uno de los compañeros del grupo.

Tuvimos conflictos al modificar nombre de clases y eliminar clases que ya no utilizaríamos

Varios conflictos con al realizar los test. Compartíamos el classpath y cuando pusheabamos se rompía para el otro.

Desarrollo de app de celular y conductor

Patrón strategy

Para el desarrollo de la funcionalidad de app del usuario. Decidimos utilizar el patrón strategy ya que el usuario debería seleccionar el modo de funcionamiento de la app (manual o automático).

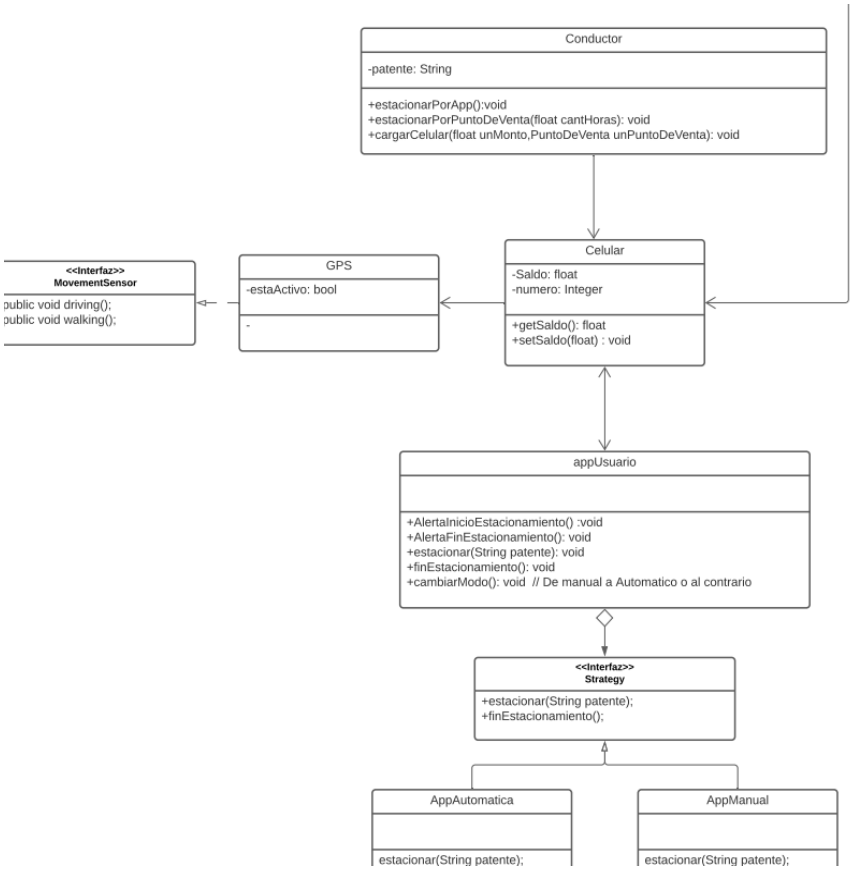
En un principio se definió utilizar una clase celular en el medio del usuario y la app de celular que se implementó con strategy. Pero después de consultas se vio innecesario el uso de esta clase.

Al final solo se decidió dejar la clase App celular que guarde todos los datos.

Otra duda que tuvimos es ver como implementar la interfaz dada por el enunciado (MovementSensor). Al realizar por primera vez el uml creímos que debería ir dentro del gps. Luego de distintas consultas se vio mejor separar ambas e implementar directamente a la appUsuario.

Los participantes del strategy:

- **Context**: Definimos la clase AppUsuario como el context
- **Estrategia**: Definimos la interfaz ModoDeApp que será usada por el contexto para invocar a la estrategia concreta.
- **Estrategia concreta**: las clases AppAutomatica y AppManual Implementa el algoritmo utilizando la interfaz definida.



UML antes de definir eliminar la clase celular.

Clase Sem

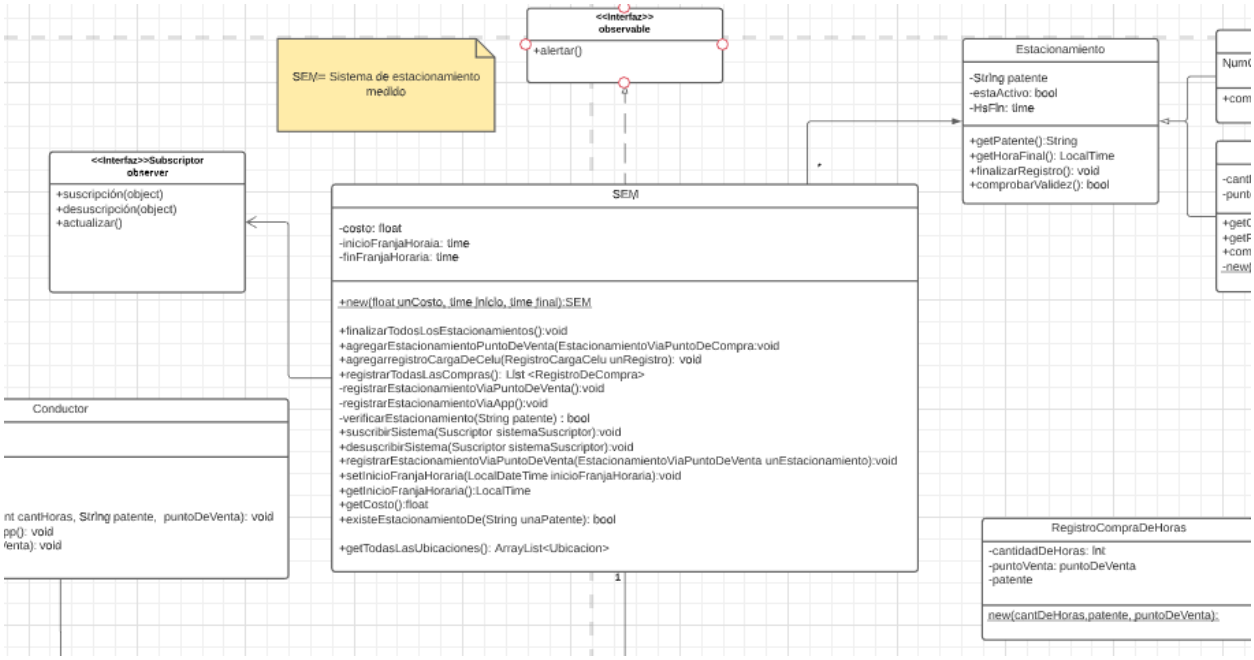
Patrón Observer

La clase Sem la utilizamos para registrar todos los cambios y registros tanto en los nuevos estacionamientos, zonas de estacionamiento, compras, suscripciones y de suscripciones del mismo.

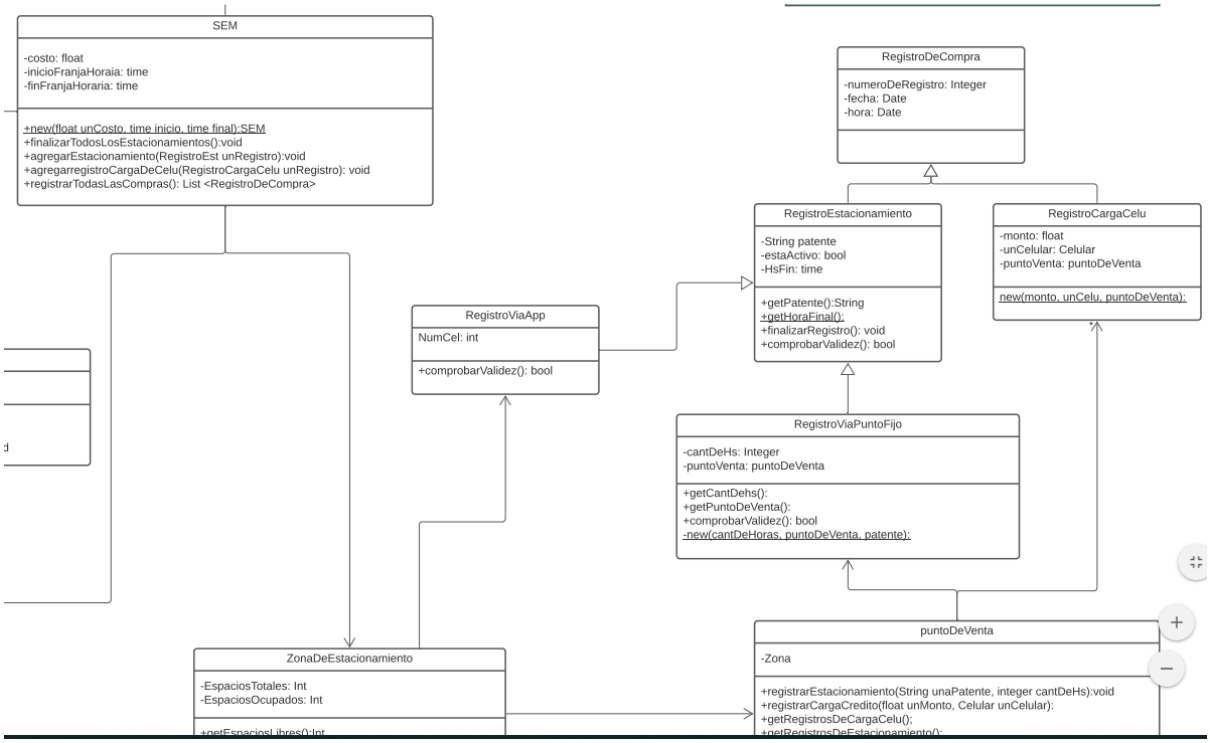
Se decidió utilizar el patrón observer ya que era necesario para generar alertas a distintas entidades o sistemas

Participantes del observer

- **Sujeto (*subject*):** Definimos a la clase SEM quien conoce a todos sus observadores.
- **Observador (*observer*):** Definimos a la clase Subcriptor, estos observan, así en caso de ser notificados de algún cambio, pueden preguntar sobre este cambio.
- **Concrete observer** : clase sistemas que estan interesados en el SEM.
- **Observador concreto:** es la clase observable



Versión final de UML con el observer.



En nuestra primera versión las clases estacionamientos eran llamadas registros. Por cuestiones de responsabilidades se modificaron.

Clase Zona de estacionamiento

Para esta clase asumimos que cada zona tiene un solo inspector a cargo de la misma.

Esta guarda los puntos de ventas, las compras por punto de venta, su inspector y ubicación. Esta información está disponible para la clase Sem.

Clase Inspector

Un inspector tiene una variable sem por necesidad de acceso a la información y la verificación de los estacionamientos.

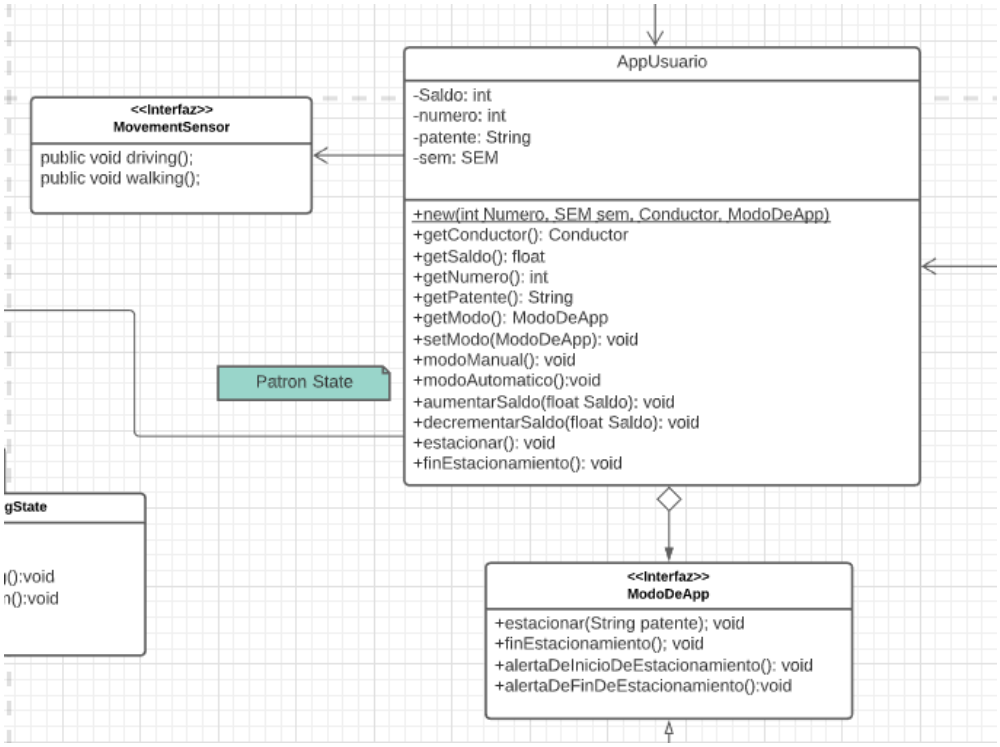
El inspector es quien se encarga de la verificación de los estacionamientos y generación de multas mediante una clase AppInspector.

Clase Punto de Venta

Se encarga de la registrar tanto las cargas de crédito para la app de usuario, los estacionamientos y compras, además de contener una zona propia.

Interfaz MovementSensor

Descartamos esta interfaz dentro del código ya que la misma influía en el coverage del proyecto. De Igual forma tuvimos en cuenta la misma en el UML



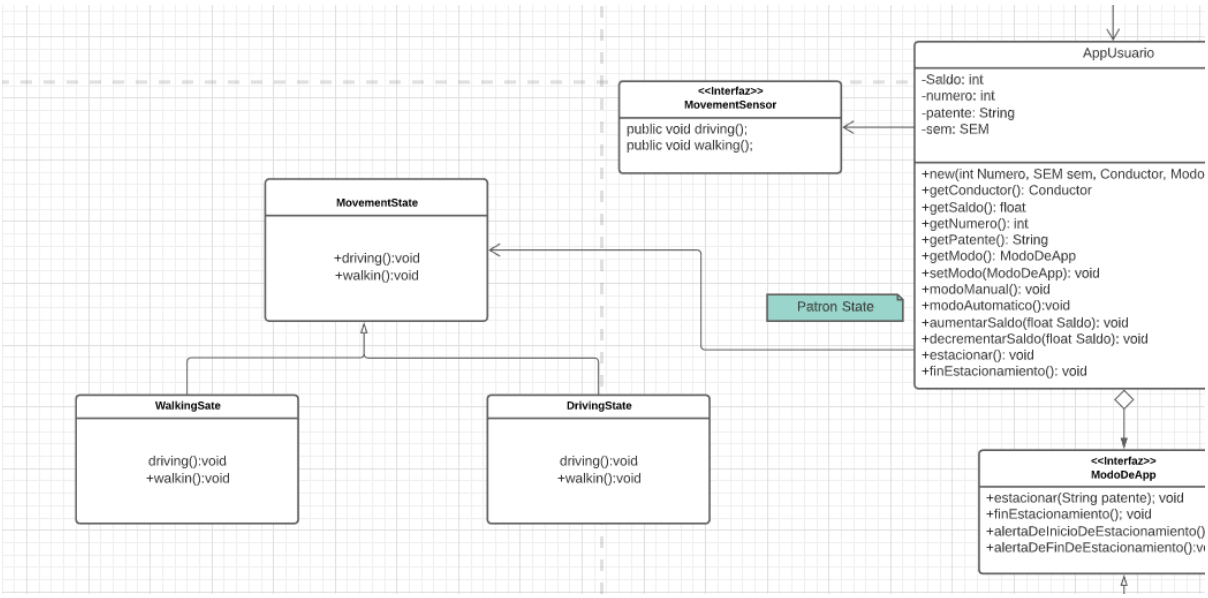
Captura del UML final del proyecto con la interfaz MovementSensor

Patrón State

Este patrón no lo implementamos en el código, pero lo reconocemos en el uml. para definir los estados Caminando y Manejando (Driving y walking).

Participantes del State

- **Context(Contexto):** Este integrante se define como la clase AppUsuario. Mantiene una instancia de ConcreteState (Estado Concreto) que define su estado actual
- **State (Estado):** Se definio la interfaz MovementState para el encapsulamiento de las responsabilidades asociadas con un estado particular de Context.
- **Subclase ConcreteState:** Las clases WalkingState y DrivingState son las subclases que implementan el comportamiento o responsabilidad de Context.



Captura del UML con el patrón State