Decisiones de diseño para el TP

# Puntos de interés

Los puntos de interés decidimos manejarlos con una herencia con una clase padre *POI* que define los métodos genéricos de un punto de interés, ya que los mismos se redefinen en cada clase hija que representa cada especificación de un punto de interés.

Cada una de las especificaciones redefine los métodos en los cuales el método no es genérico y añaden los campos que necesitan. A partir de la tercera entrega, decidimos usar el patrón Builder para cada especificación de *POI* que se justifique su uso, ya porque un objeto tiene muchos campos o porque esos campos son objetos de otras clases también definidas por nosotros (caso Rubro o Comuna). Este patrón nos facilitó la instanciación de objetos en los tests y agregó mas declaratividad a los mismos.

# Repositorio Local

El repositorio local lo creamos como una clase hija de *CollectionBasedRepo* por lo cual tuvimos que hacer que *POI* herede de la clase *Entity* para poder ingresarlos al repositorio (hecho que nos forzó a hacer concreta la clase *POI* y no abstracta como antes). En *POI* se redefinieron algunos métodos necesarios para validación heredados de Entity.

# Repositorio Externo

Para el manejo del servicio externo tuvimos que crear una clase *AdaptadorServicioExterno* que desencripte los resultados del servicio externo representado por la interfaz *interfazConsultaBancaria* (para los tests se usó un mock de la clase). Además, todos los orígenes de datos (Adaptador y Repositorio Local) implementan la interfaz *OrigenDeDatos* que define como debe ser una consulta genérica.

El funcionamiento del mismo primero fue mediante una clase Stub para el servicio externo pero el mismo no era polimórfico para distintos orígenes de datos, por lo que se decidió que el repositorio local tenga un conjunto de servicios externos, entonces si un POI no era encontrado en el repositorio local se iba a buscar a el/los repositorios externos.