TP DISEÑO 2018 - ENTREGA 0 - DECISIONES DE DISEÑO

En general, decidimos para esta entrega explicitar en el diagrama de clases todos o casi todos los atributos ya que no teníamos más requisitos que justificasen tener que priorizar la selección de elementos a comunicar. Entendemos que a medida que avance en complejidad, es importante acotar la información que se comunica a solo lo necesario para expresar dicha funcionalidad.

CATEGORIZADOR: Elegimos una especie de repositorio para tener todas nuestras instancias de categoría residencial en una lista. De este modo, cada usuario puede apuntar a uno de estos objetos y no tenemos la necesidad de generar instancias “repetidas” para cada cliente.
Evaluamos como posibilidad el uso de un dictionary pero no encontramos el beneficio práctico de este tipo de estructura.
Este categorizador tiene la responsabilidad también de a traves de recibir un consumo (tendremos que ver con las sucesivas especificaciones como se calcula el concepto de los 3 meses) poder decir a qué categoría corresponde. En este caso, también queda pendiente saber de quién es la responsabilidad de asignarle una nueva categoría al cliente.
Primero habíamos pensado métodos que permiten agregar o sacar categorías pero llegamos a la conclusión de que no lo solicita. En el caso de solicitarlos, sería muy simple de implementarlos dentro de esta clase, por el momento.
En este caso dejamos el método que asigna categoría pero estamos pendientes de tener en claro de quién es dicha responsabilidad.

CATEGORIA: Lo modelamos como una clase porque el comportamiento y los atributos son iguales en todas las categorías residenciales.
Decidimos no poner una clase superior / interfaz para englobar las categorías residenciales de las industriales porque aún no tenemos especificaciones como para argumentarlo.
Por otro lado, estas categorías se comportan como un strategy el cual delega el comportamiento al objeto que sabe resolverlo. En este caso, es medio simplificado porque todos los objetos tienen el mismo comportamiento, simplemente cambian ciertos atributos. Igualmente sigue manteniendo la flexibilidad de que se pueden agregar nuevas categorías y es extensible.
Habíamos pensado en primera instancia que las categorías se carguen desde un json pero luego decidimos ir por instanciarlos e inluirlos directamente dentro del categorizador. El dominio no específica y si suponemos usar un json estaríamos atandonos a ese source de datos cuando tal vez próximamente nos digan que es otro. En este caso, al ser simple la implementación está abierta a invertir ese tiempo en el momento que se especifique el source de dichos datos.

ADMINISTRADOR: En este caso descartamos la posibilidad de crear una clase superior / interfaz común a CLIENTE porque no tenemos detalles del comportamiento ni del dominio. El beneficio sería compartir atributos como “nombre”, “apellido”. Consideramos que el costo

de refactor es pequeño en el caso de querer implementarlo entonces esperaremos a que avance un poco la especificación para tomar esa decisión.

CLIENTE: Hoy en dia no tenemos el requerimiento de clientes industriales, por lo tanto, no consideramos necesario generar una abstracción que contemple este tipo de clientes.

DISPOSITIVO: Modelamos un dispositivo de manera genérica. Estaremos atentos a futuros requisitos que contemplen dispositivos inteligentes en la solución. Decidimos documentar en la solución el método “estaEncendido” a pesar de que por el momento es un “pasamanos” de un getter para generar menos acoplamiento con respecto a cliente. En caso de que a futuro la manera de determinar el cliente consista de un comportamiento más complejo, esa modificación no va a afectar al cliente.