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.
Si bien queda un poco tedioso inicializar las categorías dentro del categorizador, es la solución más simple ya que estamos contemplando que tendremos que refactorizar cuando se especifique un poco más del dominio. Una idea más compleja era generar un builder para que nos construya el categorizador y luego si cambiaba la manera de construirlo, simplemente modificamos el builder.
Por otro lado, pensamos en que el categorizador es candidato a un singleton (entendemos que es parecido a un WKO) porque no va a haber más de una instancia del categorizador pero no quisimos complejizar por el momento. Consideramos re-evaluarlo en futuras iteraciones.
En este caso dejamos el método que asigna categoría pero estamos pendientes de tener en claro de quién es dicha responsabilidad.
Decidimos, al final, que el categorizador se convierta en un Singleton, que sea conocido por todos para no tener que instanciarlo, ya que no es necesario tener más de una instancia de nuestro categorizador, acercándose más al concepto de repositorio.
También se decidió que el Categorizador no asigne la categoría, sino que simplemente la devuelva; delegándole entonces la responsabilidad a Cliente. Quedamos pendientes a nueva información acerca del dominio para establecer definitivamente a qué componente le corresponde recategorizar.

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 incluirlos 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.

En una instancia habíamos pensado de tener el rango de categorías en la clase categorizador en vez de tenerlo en la categoría pero esto generaba mucho acoplamiento entre ellas. Cada vez que debíamos agregar una categoría, nos implicaba modificar ambos componentes.

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.
En cuanto a sus métodos, en el total de dispositivos encendidos también se puede resolver sumando los apagados con los prendidos, pero se prefirió contar la totalidad de los dispositivos porque además de no afectar en gran medida al requerimiento, no se rompe al agregar un nuevo estado del dispositivo.
Resolvimos, en base a los requerimientos añadidos vía mail, que el Cliente entienda que puede ser recategorizado en base a su consumo. Sospechamos que es probable, en un futuro, refactorizar esto; si bien aún no hay conocimiento en profundidad acerca del modelo de negocio en este aspecto.
Decidimos que toda instancia de Cliente comience con la categoría R1, siendo esta la básica, al no tener un histórico de consumos; en vez de otorgarle el consumo en base a los dispositivos.

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.