Justificaciones de Diseño TPA – Entrega 1

Grupo 1

Diagrama de Clases

Las *clases concretas* **PersonaJuridica** y **Heladera** deben estar relacionadas, pero consideramos que es innecesario que **Heladera** conozca qué **PersonaJuridica** está a cargo de ella. Por lo tanto, **PersonaJuridica** debe conocer qué **Heladeras** tiene a cargo (si es que tuviera). Además, descartamos una relación bidireccional ya que generaría un mayor acoplamiento entre ambas clases.

Se modeló **Cuestionario** como *clase concreta* para lograr el dinamismo de **Pregunta** a responder. Cada **Pregunta** tiene una **Respuesta** que puede ser con **Opcion**es predefinidas, de elección única o de libre respuesta. **PersonaFisica** y **PersonaJuridica** conocen su **Cuestionario** y su lista de **Respuesta** a cada **Pregunta**. **Respuesta** conoce a qué **Pregunta** corresponde, pero no consideramos necesario que **Pregunta** conozca su **Respuesta** dado que a una misma **Pregunta** pueden existir infinitas **Respuesta**, pero para una **Respuesta** existe una **Pregunta**. Una relación bidireccional nos generaría mayor acoplamiento entre estas dos clases.

La *clase concreta* **DonacionDeVianda** se modeló con un único *atributo* **PersonaFisica** ya que PersonaJuridica no puede donar viandas.

Las *clases concretas* **PersonaFisica** y **PersonaJuridica**, no fueron relacionadas de ninguna forma para evitar forzar un polimorfismo que hasta esta instancia resulta innecesario (principio KISS). Además, una interface Colaborador no tendría sentido ya que **PersonaFisica** y **PersonaJuridica** no comparten comportamiento. Al mismo tiempo, evitamos el alto acoplamiento que conllevaría una herencia entre las clases Colaborador y **PersonaFisica** y **PersonaJuridica**.

Nos parece innecesario y redundante que **DonacionDeVianda** conozca en qué **Heladera** está, sabiendo que **Heladera** ya posee una lista de **Viandas**. Esto evita generar un alto acoplamiento que habría en una relación bidireccional entre ambas clases.

PuntoEstrategico se modeló como *clase concreta* para evitar la inconsistencia de datos.

Interpretamos que si **PersonaVulnerable** está en situación de calle, entonces su **Dirección** será *NULL*. De esta manera, no es necesario que tenga un atributo booleano “estaEnSituacionDeCalle”. Con la misma lógica, si **PersonaVulnerable** no tiene menores a cargo, entonces el atributo **menoresACargo** será 0.

TipoDocumento se modeló como un *enum* dado que no consideramos necesario que sea extensible en tiempo de ejecución.

Mapa Interactivo

Se eligió *Leaflet* debido a su buena documentación para realizar la implementación del mismo, lo que creemos nos dará flexibilidad. Además que también tiene a disposición variedad de plugins que podrían facilitar la incorporación de futuras funcionalidades, si fuera necesario.

A su vez, también la facilidad que provee para agregar marcadores personalizados en ubicaciones determinadas.

Además, nos pareció una gran ventaja que fuera gratis y open source.

Validador de contraseñas

Para el diseño del validador, buscamos implementar la esencia del patrón de diseño *Strategy* con una interface pero un poco modificado, ya que en nuestra clase **Validador** vamos a tener como *atributo* una lista de **formasDeValidacion**. Esta implementación nos permite que quede más modular, analizable, cohesivo y con menor acoplamiento todas las formas de validación con el validador. Además, si en un futuro quisiéramos mejorar el validador y agregarle más formas de validación, con esta implementación sería viable hacerlo.

En cuanto a cómo se informan los errores, se intentó guardar en una lista indicaciones para los distintos casos de validación que señalarían por qué falló la contraseña elegida (longitud, caracteres no válidos, etc) para informar en un futuro al usuario.