**Justificaciones de las decisiones de diseño tomadas**

* Entrega Modelado en Objetos Parte I: Puesto a punto del entorno de desarrollo y primera iteración del diseño

En primer lugar, fuimos identificando y guardando información relevante relacionada al Estado Interno de cada clase, la cual consideramos que vamos a necesitar en un futuro para desarrollar el sistema.

En Servicios Públicos, creamos una sola clase que represente ferrocarriles y subterráneos porque no identificamos nada que los diferencie a la hora de utilizarlos en el sistema en base a la funcionalidad.

En Servicios, separamos los Medios de Elevación de los Baños ya que, tenemos que guardar información propia de cada una de esas clases. También diseñamos una superclase abstracta llamada Servicio para poder tratar a ambas clases de manera polimórfica y guardar el atributo que tienen en común (Estado). Por esto mismo, decidimos crear una clase abstracta y no una interfaz.

En Comunidades y Usuarios, decidimos crear una lista de Administradores como atributo perteneciente a la clase Comunidad porque si nosotros le asignamos un rol (como estado interno) a la clase Usuario íbamos a tener muchos problemas ya que hay usuarios que pueden tener un rol distinto según la comunidad a la que pertenezcan. Además esto mismo genera tener una relación de asociación doble, lo cual conlleva a tener más acoplamiento entre esas dos clases.

Utilizamos una relación de agregación entre las clases Comunidad y Miembro, ya que las comunidades necesitan a un miembro para existir, creemos que no debe haber comunidades sin miembros, pero los miembros pueden existir sin necesidad de pertenecer a una comunidad.

Estas dos clases, por el momento, solo contienen atributos debido a que no identificamos ninguna funcionalidad (método) necesaria para los requerimientos dados de esta entrega.

Para verificar si una contraseña es válida, creamos una clase Verificador la cual conoce y utiliza distintas clases de condición (Longitud, Credencial y Top 10000). Estas condiciones se basan en el patrón Strategy, donde consideramos cada condición como una estrategia, ya que sino cada vez que queramos agregar una nueva condición deberíamos modificar nuestra clase Verificador la cual tendría toda la responsabilidad, generando poca cohesión ya que se encargaría de hacer muchas cosas.