Decisiones de Diseño - TPADDS

Entrega 1

**Colaborador**

Para los colaboradores pensamos en diseñarlo como una interfaz donde las Personas Humanas y las Personas Jurídicas heredan comportamiento en común además de tener sus atributos propios. Decidimos esto ya que pensamos que ambas entidades van a ser fijas y no tendría sentido que en algún momento necesiten cambiar su identidad de Persona Humana a Persona Jurídica o viceversa. Para la carga de datos adicionales, creamos una clase Formulario, la cual contiene una lista de Preguntas, modeladas con su respectiva clase. Cada pregunta tiene como atributos dos Strings, uno con el contenido de la pregunta, y el restante con la respuesta. En conclusión, para la carga de datos adicionales a los fijos de cada colaborador, se implementa un formulario y se agregan las preguntas que se deseen. La anterior abstracción sobre el funcionamiento del formulario, la pensamos para garantizar flexibilidad en el manejo de los campos que cada ONG requiera al registrar un nuevo colaborador.

**Colaboraciones**

En el caso de las colaboraciones planteamos una Interfaz Colaboración, donde cada una de las Colaboraciones la implementan, incluyendo en cada uno de las 4 Colaboraciones { DonarDinero, DonarVianda, DistribuirVianda, HacerseCargoHeladera} el metodo aplicarColaboracion() favoreciendo el uso del polimorfismo en POO. Es importante aclarar que en cada Colaboración se incluye la etiqueta @Override para que cada Colaboración implemente el método a su conveniencia. También en cada Colaboración se incluyeron sus respectivos atributos.

**Viandas**

Para las Viandas pensamos en modelar una Clase con sus respectivos atributos para que puedan ser instanciadas la cantidad que el usuario desee. Cabe aclarar que a parte de sus atributos de tipo Heladera, Colaborador, Integer, Float pensamos en implementar los tipos Estado que consiste en un Enum { ENTREGADA, NO\_ENTREGADA} y el tipo Comida, que si bien lo modelamos como un Enum, ante la falta de información por parte del enunciado, creemos que en un futuro esto podría transformarse en una clase.

**Persona Vulnerable**

Para la entidad Persona Vulnerable decidimos crear una clase con sus respectivos atributos; nombre, fechaNacimiento, fechaIngreso, situacionDeCalle, direccion, tipoDocumento, numeroDocumento, menoresACargo y cantidadMenosACargo. Donde la mayoria de los mismos son tipos de datos propios de Java, salvo el tipo documento, que lo decidimos modelar como un Enum, ya que creemos que en un futuro, la persona vulnerable va a tener un comportamiento dependiendo de este campo.

**Heladera**

Para la entidad Heladera, pensamos en atributos de tipo String para su nombreSignificativo y para su dirección. Luego para su cantidadDeViandas en un simple Integer y finalmente para su puntoEstrategico pensamos en modelar una clase PuntoEstratégico pensando en una futura implementación con una aplicación externa, con sus respectivos atributos de longitud y latitud.

**Contraseñas**

Para implementar controles contra contraseñas débiles en Java y verificar si una nueva contraseña se encuentra en la lista de las 10,000 peores contraseñas, implementamos la siguiente solución:

* Descargamos la lista de las 10,000 peores contraseñas (Del repositorio de SecLists, un proyecto incluido oficialmente por OWASP IOT)
* Leemos la lista y la almacenamos en un HashSet, que para este caso en particular hace más eficiente la búsqueda
* Al momento de que un usuario ingrese una nueva contraseña, verificaremos si esa contraseña está en el HashSet de contraseñas débiles. Realizaremos la misma operación al cambiar la credencial por una nueva.

Para alinear la política de longitud, complejidad y rotación de contraseñas con las recomendaciones de la Sección 5.1.1.2 de la Guía NIST 800-63, se implementó la clase ValidarPassword.

* La longitud mínima de la contraseña se establece en 8 caracteres, como recomienda NIST.
* La complejidad de la contraseña se verifica utilizando una expresión regular que requiere al menos un número y una letra en la contraseña.
* El método esPassValida verifica tanto la longitud como la complejidad de la contraseña.
* El metodo esPassDebil unifica el criterio de validez con la presencia de la credencial en la lista de contraseñas más comunes

Entrega 2

**Registro de Persona Vulnerable**

El registro de persona vulnerable se agregó como una nueva colaboración, la cual contiene los atributos suficientes para registrar a la persona vulnerable. Además contiene una referencia a la tarjeta a repartir. La tarjeta fue modelada como una clase, y contiene referencia hacia la persona dueña, su código alfanumérico y una lista de usos. Dichos usos fueron modelados como una clase aparte, garantizando el registro de la fecha del uso y la heladera en cuestión. Los métodos de la clase tarjeta garantizan la validación de requisitos para que esta sea usada por la persona dueña.

**Seguimiento de Colaboradores**

Para este requerimiento se implementaron tres clases, cada una diseñada para garantizar un funcionamiento apropiado y una cohesión óptima. La clase Reporte contiene la información del colaborador, el puntaje obtenido y la fecha de emisión del reporte. Esta separación asegura que cada Reporte mantenga una responsabilidad clara y específica. Luego para la creación de estos, se implementó la clase GeneradorReportes, la cual dada una lista de colaboradores, itera sobre la misma creando el reporte de cada uno. Para el cálculo de los puntajes, se agregó la clase CalculadorReporte, la cual dado un colaborador y su lista de colaboraciones realizadas, calcula los respectivos puntajes obtenidos por cada colaboración realizada separando la lógica de cálculo en una clase específica para mayor cohesión, permitiendo modificaciones y ajustes en la fórmula de cálculo sin afectar otras partes.

**Carga masiva de Colaboradores**

Para garantizar una correcta cohesión abocada a este requerimiento, se decidió utilizar una interfaz conformada por 3 componentes {ColaboradorCSV, CargaColaboradoresCSV, LectorCoeficientes}. En cuanto a la primera clase mencionada, se pensó en un tipo de colaborador que contenga los datos provistos en los archivos recibidos. Esta abstracción se dio al tener menos datos de los suficientes para crear un colaborador. Por lo que al leer cada línea del archivo provisto, se creaba una instancia de ColaboradorCSV, y junto con la lista de colaboradores ya en sistema, se realizaba una búsqueda y correspondiente carga de colaboración, o dado el caso la carga de un nuevo colaborador humano en el sistema.

**Sensor de Peso**

En la implementación del sensor de peso, se pensó en implementar un patrón Adapter. Para ello se creó una API conformada por los siguientes componentes:

En primer lugar se creó una clase Reading, para representar el formato de lectura de los pesajes, según lo que el cliente proveerá en un futuro. Luego se implementó una interface SensorPeso, la cual contiene un el método ‘obtenerReadingPeso’. A esto le siguió la inclusión de una clase AdapterPeso, la cual hereda de SensorPeso, y cuyos atributos son:

· Heladera: referencia hacia la heladera que utiliza el sensor de peso.

· SensorPesoTercero: referencia hacia la API provista por el cliente.

· pesoYUnidad: referencia hacia el tipo de dato que devuelve la lectura del peso.

A estos atributos se le adhieren el método heredado de la interface, el cual obtiene del sensor proveido por el cliente, un tipo de dato Reading, y del cual saca el valor del peso con el método ‘getPeso’. El último método de la clase es ‘calcularEstadoHeladera’, el cual obtiene de la heladera su capacidad máxima, y con el valor de peso ya obtenido, calcula el porcentaje de llenado y devuelve el estado de la misma. Dicho estado fue representado como un Enum ‘EstadoCapacidadHeladera’.

**Sensor de Temperatura**

En cuanto a la implementación del sensor de temperatura, se utilizó el mismo patrón que en el sensor de peso, llevando con ello consistencia con respecto al manejo de los sensores. En consiguiente se conformó la API de la siguiente manera:

Al igual que en el sensor de peso, se utilizó una clase conectora con la API del cliente, llamada ‘TemperatureSensorTercero’. Luego como clases que adaptan el comportamiento, se implementó una interface ‘TemperatureSensor’, la cual contiene los métodos ‘comunicarTemperatura’ y ‘realizarLectura’. A ello se le agregó la implementación de la clase ‘SensorTemperatura’, cuyos atributos son:

· heladera: referencia hacia la heladera del sensor.

· temperaturaMaxima: se setea de acuerdo al valor propio de la heladera.

· temperaturaMinima: se setea de acuerdo al valor propio de la heladera.

· Lecturas: lista de lecturas de temperatura.

· sensorTercero: referencia hacia la API provista por el cliente.

· Scheduler: referencia hacia el componente que realiza el método periódico en el tiempo.

En cuanto al scheduler, se importó la biblioteca ‘java.util.concurrent’, invocando al ejecutor ScheduledExecutorService usándolo como tipo de dato del scheduler. La clase comentada hereda de la interface mencionada anteriormente. De los métodos mencionados, obtenerTemperatura se encarga de traer la lectura del artefacto provisto por el cliente. Luego se implementaron los métodos ‘agregarLectura’, el cual recibe por parámetro una temperatura y la agrega a una lista de lecturas, la cual es utilizada en el método ‘necesitaAtencion’, quien dada la lógica del enunciado, evalúa si la heladera necesita atención o no. Por último se definió el método ‘realizarLecturas’, cuyo propósito es el de realizar lecturas cada 5 minutos, agregarlas a la lista de lecturas y evaluar si la heladera necesita atención.

**Análisis crítico de los requerimientos**

b. El Sistema rinde cuenta a:

• La ONG para la que va dirigido

• Colaboradores

• Personas en situación vulnerable

El sistema no rinde cuenta:

• Desarrolladores

c. Si, hay partes que están interesadas y no están tenidas en cuenta, una de las partes que no está tenida en cuenta en el diseño son las personas vulnerables. Teniendo en cuenta que el propósito del sistema es brindar ayuda a las mismas, consideramos que si deberían ser tenidas en cuenta en este diseño. Otra de las partes que no está tenida en cuenta son los colaboradores que son los encargados de hacer las distintas donaciones, y creemos que también deberían estarlo, ya que podrían ayudar en el diseño de las colaboración que ellos hacen.

d. Nosotros pensamos que por ahora los requerimientos están bien, aunque sí podrían ser modificados teniendo en cuenta a los colaboradores y a las personas vulnerables. Por más que nosotros pensemos que estamos haciéndolo de la mejor manera para que ellos lo utilicen, pueden llegar a necesitar cosas distintas en el día a día las cuales podemos no tener en cuenta a la hora del diseño del sistema ya que no sería utilizado por nosotros mismos.

Entrega 3

**Colaboradores y Tarjetas**

Para empezar a modelar esta parte, se pensó en agregar una nueva clase TarjetaColaborador, y adecuarla a la tarjeta ya existente TarjetaPersonaVulnerable. Ambas clases implementan un interface Tarjeta, para garantizar el polimorfismo con el método usarTarjeta, el cual recibe una heladera como parámetro.   
Dentro del uso de la tarjeta en esta nueva clase, se realiza una apertura de heladera. Como para ello se debe haber hecho una solicitud de apertura anteriormente, se decidió implementar una clase SolicitudApertura.   
Para notificar a la heladera que una tarjeta se encamina a realizar una apertura, se implementó un patrón adapter con la interface proveída por el cliente. En el mismo adapter se implementó una lógica propia para implementar el permiso de las tarjetas luego de generar la solicitud. La lógica se decidió implementar directamente en el adapter, para garantizar testeabilidad y desacoplamiento con la heladera en sí.  
En cuanto a la lógica mencionada, se decidió que el adapter contenga una lista de tipo String, conteniendo los números de tarjeta que van llegando. Esta abstracción nos permite validar cuando el colaborador hace uso de la tarjeta, si se le permite acceso a la heladera, verificando que el número de la misma, este contenido en la lista del sensor controladorDeAcceso.   
Luego cuando el colaborador fehacientemente realiza la apertura previamente solicitada, se registra una AperturaHeladera, otra clase con los detalles propios de la apertura. Tanto para la solicitud como para la apertura en sí se registran en un repositorio de movimientos, así garantizamos trazabilidad.

**Incidentes:**

Para la lógica de los incidentes, se creó una clase abstracta Incidente, la cual al generarse llama al técnico más cercano con el método avisarTecnico(). A partir de esta clase, diseñamos los tipos de incidentes que puede tener una heladera, los cuales heredan de la clase abstracta antes mencionada. Esta abstracción permite que haya un mejor control frente a los incidentes, tratándolos polimórficamente. Cada vez que se genera un incidente, este se carga en un repositorio, para así garantizar la trazabilidad de los sucesos.   
Con respecto a las fallas técnicas, estas pueden ser generadas por los colaboradores, a diferencia de las alertas que son generadas por la heladera.  
Para la arquitectura del técnico de heladera se decidió diseñar como una clase nueva, que contenga la localización en que se encuentra en ese momento, para permitir que una vez generado un incidente, se lo pueda llamar.

**Notificador:**

Para la parte de Notificar, creamos una clase Notificador, que es el encargado de notificar a cada Colaborador por el medio correspondiente, para esto, el mismo tiene tipos de clases, WhatsApp Sender, Telegram Sender y Mail Sender, donde los dos primeros son implementados con un Adapter, necesario para comunicarnos con la entidad externa (Instant Message) que realmente tiene el comportamiento para la comunicación. Por otro lado, el MailSender se implementa vía una API (JakartaMail). Donde la consumimos para que se encargue del envío de mails.

Después creamos la clase Notificación, que guarda el mensaje y el Colaborador. Por lo tanto, el notificador usa esta clase y la envía hacia los distintos colaboradores.

**Suscripciones:**

Para la parte de las suscripciones, para el caso del sensor de la cantidad de viandas creamos un Adapter el cual es el encargado de conectar al manager de cantidad de viandas que es el encargado de notificar a los colaboradores en caso de que sea necesario teniendo en cuenta la cantidad de viandas a la que hay que notificarlos según lo que elijan, por otro lado tenemos un manager de incidentes el cual se encarga de notificar a los interesados de esa heladera en caso de que se produzca un incidente, para la elección de las heladeras a las que se deben llevar las viandas de la heladera dañada se creó una interfaz criterioSugerencia la cual tiene un metodo llamado generarHeladerasParaTransferir(Heladera heladera) el cual se encarga de generar las heladeras correspondientes según su criterio, este criterio está implementado en sus respectivas clases: SugerenciaMasProximas, SugerenciaVacias, SugerenciaAlAzar.