Cuidándonos - Justificaciones de Diseño

1. De las personas nos interesa saber: nombre y apellido, dirección, edad y sexo.
2. Para que una persona sea cuidadora de otra deberá tener instalada la aplicación; o sea que al menos deberá ser un usuario pasivo. Se considera usuario activo a aquel que solicita los acompañamientos.

Si bien originalmente contemplamos la opción de modelar a las personas como una clase y utilizar composición creando dos clases más, Cuidador y Transeúnte, decidimos optar por una única clase ***Persona*** ya que nos pareció que no era necesario hacer dicha distinción entre ellas. De esta forma todas las personas pueden ser tanto cuidadores como transeúntes al mismo tiempo sin tener que transicionar de una a otra.

Decidimos modelar el ***sexo*** de la persona como un enumerado ya que nos pareció lo suficientemente amplio para cumplir con el requerimiento solicitado y no generar sobrediseño. Por otro lado, optamos por crear una clase ***Dirección*** y no usar un String para la misma debido a que al ser de esa forma es mucho más probable que se cometan errores. Al ser una clase con atributos específicos podemos facilitar su uso, evitando confusiones, dándole a nuestro diseño robustez y mantenibilidad.

Por último, decidimos agregar los atributos ***numeroCelular*** y ***reaccion***, sobre los cuáles ampliaremos más adelante, y un atributo ***usuario*** de tipo ***Usuario*** (clase concreta). Si bien no estaba pedido explícitamente en la consigna, nos pareció lógico que de cada persona se guarde un ***username*** y ***password*** teniendo en cuenta el dominio.

3. Cada vez que un usuario quiera ir hacia un destino, deberá especificar la dirección exacta donde se encuentra actualmente y la del destino final; además de escoger quiénes serán sus cuidadores (puede haber un solo cuidador). Una vez especificados estos datos, se deberá presionar el botón de confirmar cuidadores. Los cuidadores seleccionados por el transeúnte serán notificados y deberán aceptar o rechazar el cuidado.

Para esto se creó una clase ***Viaje***, la cual cuenta con los siguientes atributos:

- Un ***transeúnte***, de tipo ***Persona***.
- Una lista de ***ciudadoresElegidos***, siendo éstos los que el transeúnte eligió, previo a que decidan o no hacerse cargo del cuidado.
- Una lista de ***ciudadoresQueAceptaron***, lo cuál sería una reducción (o no) de la lista anterior en base a los cuidadores que efectivamente aceptaron la responsabilidad de acompañar a la persona durante su viaje. Tener esta lista nos permite que el sistema pueda luego enviar la información relevante respecto del recorrido del usuario SOLO a los cuidadores que están haciendo un seguimiento del transeúnte.
- Un ***origen*** y un ***destino***, ambos de tipo ***Dirección***.
- Un booleano ***enCurso***, para saber si la persona finalizó su viaje o no, lo cual nos va a servir para que el sistema sepa si el usuario puede recibir notificaciones o no por temas de seguridad.
- La hora de inicio (***horaInicio***) para que posteriormente el sistema pueda saber si es necesario emitir alguna alerta en caso de que la persona se exceda del tiempo estipulado del viaje.

- El ***tiempoDeDemora***, siendo este el tiempo que tardará el transeúnte desde que comienza su recorrido hasta que lo finaliza (en el próximo punto se explica cómo fue modelada la solución para poder realizar dicho cálculo).

Consideramos que todo lo relacionado con *"presionar botones"* no es responsabilidad de esta capa y no es necesario ni correcto reflejarlo en el diagrama de clases.

Por otro lado, si bien las notificaciones particulares mencionadas en esta parte de la consigna tampoco forman parte de algo que esté dentro de nuestro alcance o responsabilidad, decidimos crear las clases ***Notificador*** y ***Notificación*** para cubrir otros requerimientos que explicaremos más adelante.

4. Si al menos un cuidador acepta la responsabilidad durante el trayecto, al transeúnte se le habilitará el botón de "comenzar". Al ser presionado este botón, el sistema deberá calcular el tiempo de demora aproximado y volverle a notificar a sus cuidadores. La distancia (en metros) entre dos direcciones será calculada por "Distance Matrix API" de Google, cuyo sistema nos brinda una interface REST.

Para calcular la demora del viaje en minutos decidimos modelar la clase ***CalculadoraTiempoDemora***, la cual posee un método *calcularTiempoEntre()* que recibe tanto el ***origen*** como el ***destino***, es decir que calculará el tiempo entre dos direcciones.

Para poder utilizar la API de Google decidimos usar el ***patrón Adapter*** ya que, si bien sabemos que nos devolverá la distancia entre dos direcciones, no conocemos cómo realizará el cálculo. La interfaz ***AdapterCalculadoraDistancia*** nos permite seguir adelante con nuestra implementación simplemente conociendo la responsabilidad de la API, sin preocuparnos por su comportamiento específico.

CalculadoraTiempoDemora cuenta con un atributo ***adapter*** para poder utilizar el método perteneciente a la interfaz, y así, calcular el tiempo de demora en base a la distancia que deberá recorrer la persona.

5. Durante todo el recorrido, el sistema no deberá enviar notificaciones al transeúnte (por motivos de seguridad), ya que el mismo estará en movimiento.
6. Una vez que el transeúnte llegue a su destino, deberá presionar el botón "llegué bien!". El sistema deberá volver a habilitar las notificaciones, ya que se considera que no hay peligro alguno, y se deberá volver a notificar a sus cuidadores con esta situación.

Como mencionamos anteriormente, que el sistema envíe notificaciones o no en este caso está fuera de nuestra responsabilidad, así como también lo relacionado con presionar botones.

7. Si algo malo sucede, el sistema deberá darse cuenta de esta situación por el tiempo aproximado que calculó. El mismo va a reaccionar frente a este incidente según lo que haya configurado el usuario:
- ***Enviar un mensaje de alerta a sus cuidadores***
- ***Realizar una llamada automática a la policía***

- ***Realizar una llamada al celular del usuario***
- ***Esperar N minutos para ver si es una falsa alarma (los minutos deben ser parametrizables).***

Utilizando el ***patrón Strategy***, modelamos las distintas formas de reaccionar como una interfaz (***TipoReaccion***) ya que las clases que la implementan (***NotificarCuidadores***, ***LlamarPolicia***, ***LlamarUsuario***, ***EsperarNMinutos***) comparten un mismo método *reaccionar()* pero cada una tiene su propio comportamiento. Consideramos que no sería correcto modelar esto con un patrón State debido a que las clases no se conocen entre sí y no hay transición alguna entre las mismas.
El método *reaccionar()* recibe por parámetro un viaje, para poder obtener así la información necesaria y proceder según haya configurado el usuario.

- Decidimos que tanto para *notificar a los cuidadores* como para *llamar al usuario* se haga a través de su teléfono celular, es por eso que el mismo se guarda en el atributo ***numeroCelular*** de la clase ***Persona***.
- ***LlamarPolicia*** deberá realizar dicha acción, aunque no necesita ninguna información adicional sobre el viaje en sí.
- ***EsperarNMinutos*** cuenta con un atributo ***minutosAEsperar*** para que el usuario pueda configurar la cantidad de tiempo que desea aguardar hasta verificar si hubo una falsa alarma, y una ***reaccion*** (del tipo ***TipoReaccion***) al igual que la ***Persona***. Esto permite que finalizado el tiempo de espera elegido se pueda proceder con alguna otra de las reacciones ante una emergencia.

Se debe considerar que pueden surgir nuevas formas de reaccionar frente a un incidente y que el usuario puede cambiar esta configuración cuantas veces quiera.

Se cumple con el requerimiento solicitado al tener en la ***Persona*** un atributo ***reaccion*** (del tipo ***TipoReaccion***), el cual permite que la misma elija según su preferencia de qué modo va a reaccionar frente a un posible incidente, pudiéndose modificar si así lo desea.

Ahora un transeúnte también podrá escoger un destino con varias paradas. Para esto, se deberá especificar la dirección exacta de cada destino y el orden en el que se recorrerán. Además, el usuario deberá especificar si se detendrá N minutos en cada parada, o si irá avisando punto a punto su estado de "salud" (si llegó bien).
Si se especifica que se va a detener en cada parada, entonces el sistema deberá ir calculando las demoras aproximadas por secciones (demora de A->B, demora B->C, etc.); caso contrario, se deberá hacer un cálculo aproximado total.

Para poder realizar esto decidimos crear una nueva clase ***Destino***, la cual contiene una *dirección* y el *tiempo que la persona desea detenerse en dicha parada*. A su vez el atributo ***destino*** de la clase ***Viaje*** ya no es más una ***Dirección***, sino una ***lista de Destinos***.
Esto nos da cierta flexibilidad y permite cumplir con el nuevo requerimiento contemplando todos los casos: *una persona puede elegir uno o más destinos, y puede optar o no por detenerse en cada uno de ellos por una cantidad de minutos determinada (si el tiempo es 0, es porque no desea detenerse)*.
Para hacer los cálculos del tiempo que tardará la persona en hacer su recorrido optamos por agregar una nueva clase ***CalculadoraTiempoTotal*** *(también optamos cambiar el nombre de la clase CalculadoraTiempoDemora por* ***CalculadoraTiempoParcial****, ya que nos*

pareció más coherente con lo pedido en este caso), la cual cuenta con un método *calcularTiempoTotal()* que recibe por parámetro tanto el origen como la lista de destinos. Utilizando la calculadora de tiempos parciales, como se muestra en el *pseudocódigo* adjunto en el archivo .txt, permite calcular el tiempo total del viaje contemplando que la persona se detenga o no en la/las paradas que realice sin necesidad de pasar atributos adicionales ni modificar los métodos ya existentes.