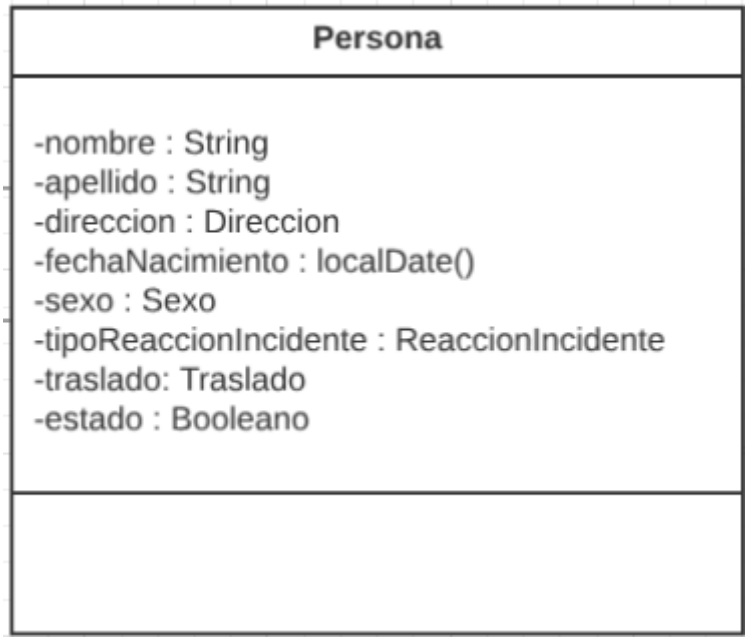
DECISIONES DE DISEÑO

PERSONA

Las personas comparten nombre, apellido, dirección, edad y sexo:
Optamos por la opción de pensar a Cuidadores y personas a cuidar como una única clase Persona.

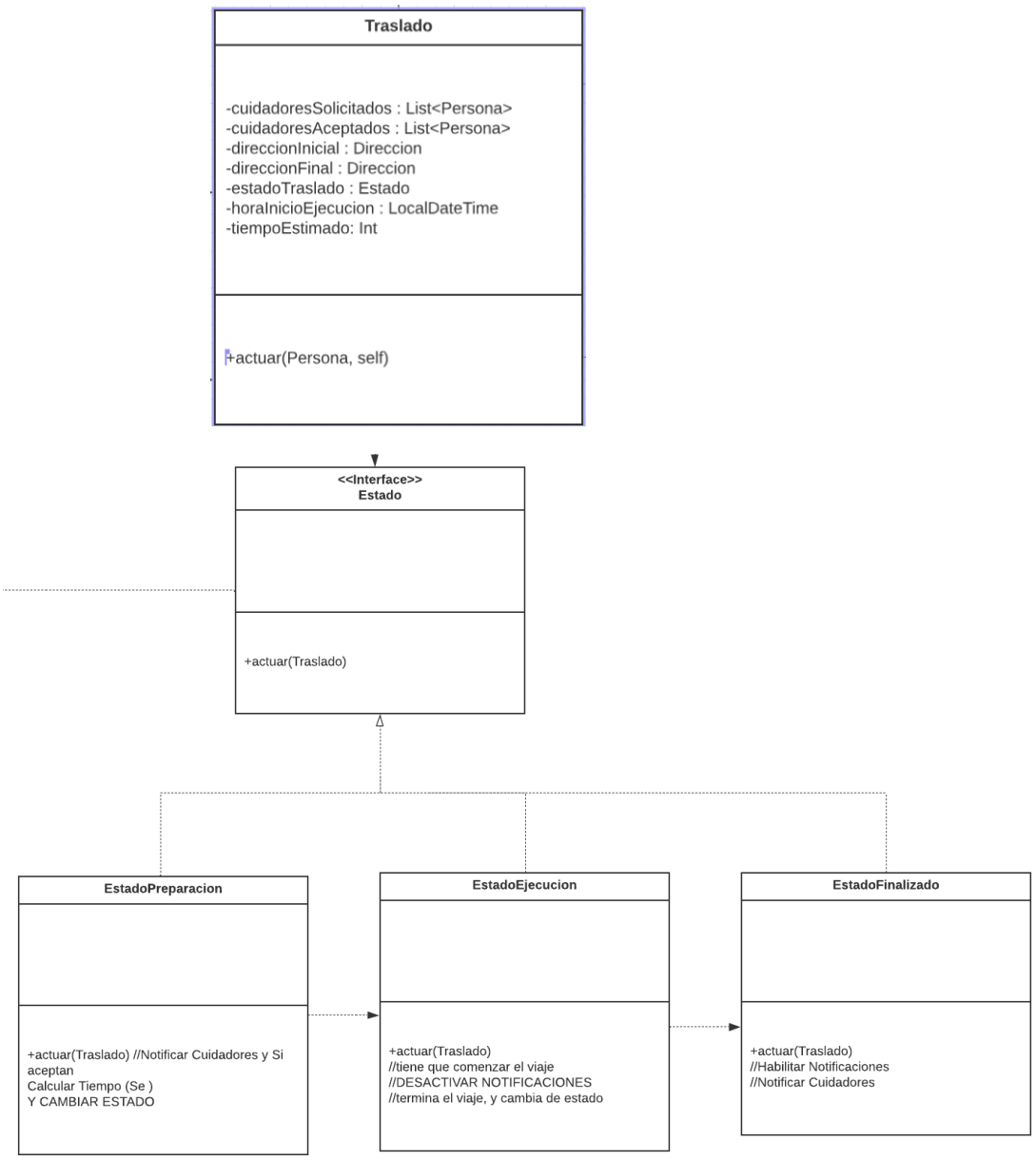


La edad se modela como fechaNacimiento → la edad se calcula.

Las personas tienen el atributo “estado”, el cual va cambiando en tiempo de ejecución. Nosotros interpretamos que, ni bien la persona descarga la aplicación, pasa a ser persona pasiva. Esto significa, que puede ser solicitado para ser cuidador. La única manera de pasar a ser activo, es solicitando un traslado. Por lo tanto, a la hora de elegir los cuidadores para un traslado, el sistema deberá filtrar la lista de personas, de modo tal que muestre a las del tipo pasivo.

TRASLADO

Se crea una clase Traslado, que tendrá varios atributos como un destino (una Dirección), un origen (otra Dirección), los cuidadores (lista de Personas), una fecha y hora de inicio de viaje, una duración estimada en minutos y un atributo que indica el estado del traslado, mediante por el cual desarrollamos un Patrón State:



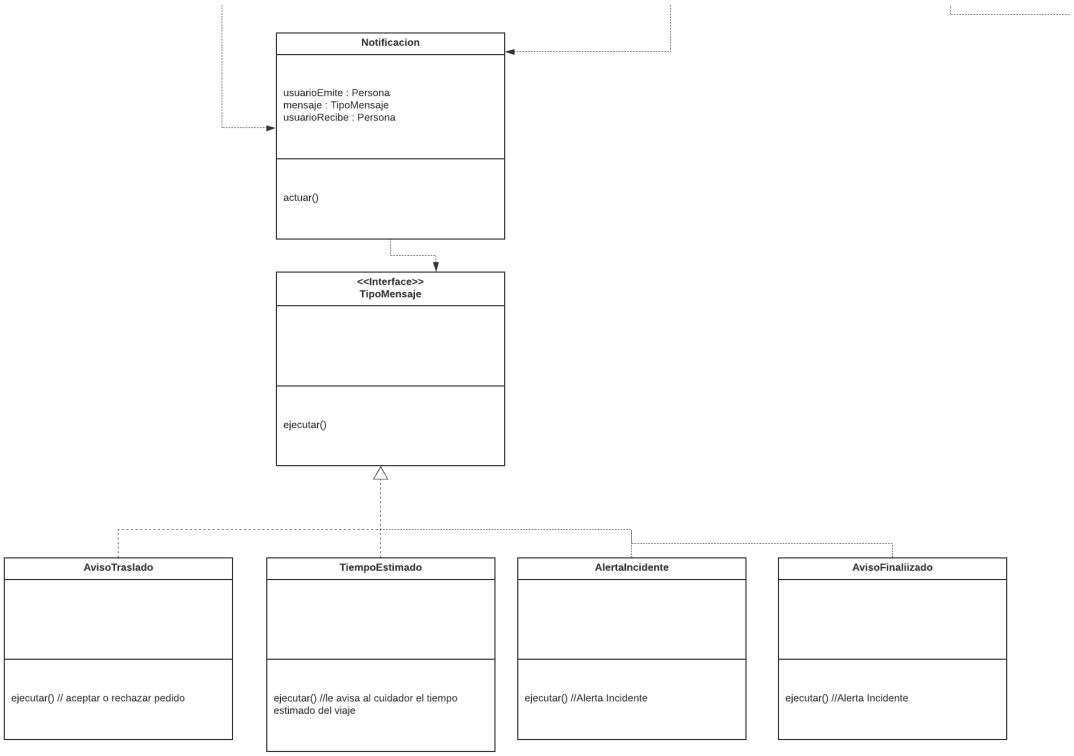
Cuando el traslado está en estado de preparación, lo primero que hace es notificar a los cuidadores. Si alguno acepta, habilita a la persona a comenzar el viaje. Se hace el cálculo del tiempo de viaje, se le notifica a los cuidadores y se realiza el cambio de estado a EstadoEjecución.

En éste estado, desactiva las notificaciones del transeúnte. La persona deberá apretar el botón de que llegó correctamente para realizar el cambio de estado a Finalizado. Si pasado el tiempo, ésto no ocurre, teniendo en cuenta la configuración de la persona, el sistema reaccionará ante el incidente.

En el último estado, EstadoFinalizado, se le vuelven a habilitar las notificaciones al usuario y se les notifica a los cuidadores que terminó el viaje.

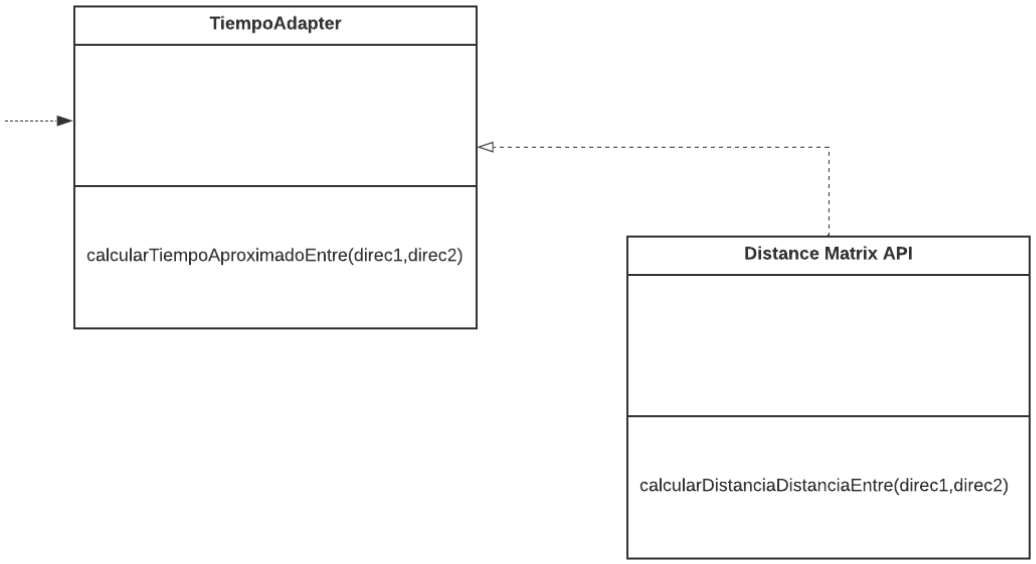
NOTIFICACIONES

La clase tendrá como atributos a un emisor, a un receptor y a un TipoMensaje, a partir de la implementación del Patrón Strategy, se podrá enviar un tipo de mensaje u otro en base a la notificación que se deba enviar en ese momento.



Tiempo Estimado

Se crea una clase TiempoAdapter, que tendrá un método calcularTiempoAproximadoEntre que recibe como parámetro dos Direcciones y devuelve un entero que representa al tiempo en minutos que se tarda en recorrer la distancia entre esas dos direcciones.
El enunciado pide que usemos “Distance Matrix API” de Google para calcular la distancia, creamos una clase DistanceMatrixAPI que use TiempoAdapter.



Reacciones a Incidentes

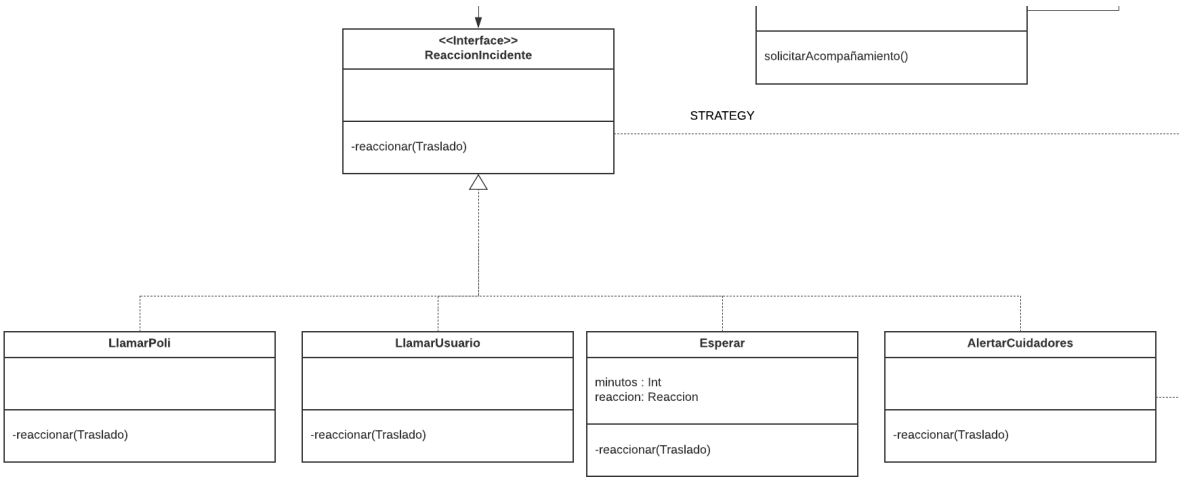
Para implementar este requerimiento, decidimos agregar el atributo tipoReaccionIncidente de tipo ReaccionIncidente y utilizar el Patrón Strategy.
Esa ReaccionIncidente será una interfaz, la cual tendrá con un método reaccionar() que recibirá como parámetro un Traslado, al haber varias formas de reaccionar frente al incidente, se crean varias clases que implementarán la interfaz, cada una con su método reaccionar() correspondiente.

AlertarCuidadores:
Recibe el traslado y de ahí extrae sus cuidadores, para enviarle una notificación de alerta a cada uno.
La clase AlertarCuidadores usará la clase Notificacion

LlamarPoli:
Llamará a la policía, ignorando el Traslado que se recibe como parámetro en el método reaccionar().

LlamarUsuario:
Se debería cargar un número de teléfono en la clase Persona, o en la clase Traslado para poder realizar esta función.

Esperar
Esta clase tendrá un atributo de tiempo a esperar, luego de que transcurra, ejecutará un atributo reacción de tipo ReaccionIncidente.


<<Interface>>
ReaccionIncidente
-reaccionar(Traslado)
solicitarAcompañamiento()
STRATEGY
LlamarPoli
-reaccionar(Traslado)
LlamarUsuario
-reaccionar(Traslado)
Esperar
minutos : Int
reaccion: Reaccion
-reaccionar(Traslado)
AlertarCuidadores
-reaccionar(Traslado)