



UTN.BA

UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Trabajo Práctico en Parejas

“Cuidándonos”


para

Diseño de Sistemas

Preparado por

Integrantes	
Nombre y Apellido	Legajo
Alvarez Kalustian, Sahira Aylén.	175.584-5
Guardines, Brenda.	172.658-4

Modelado de Dominio

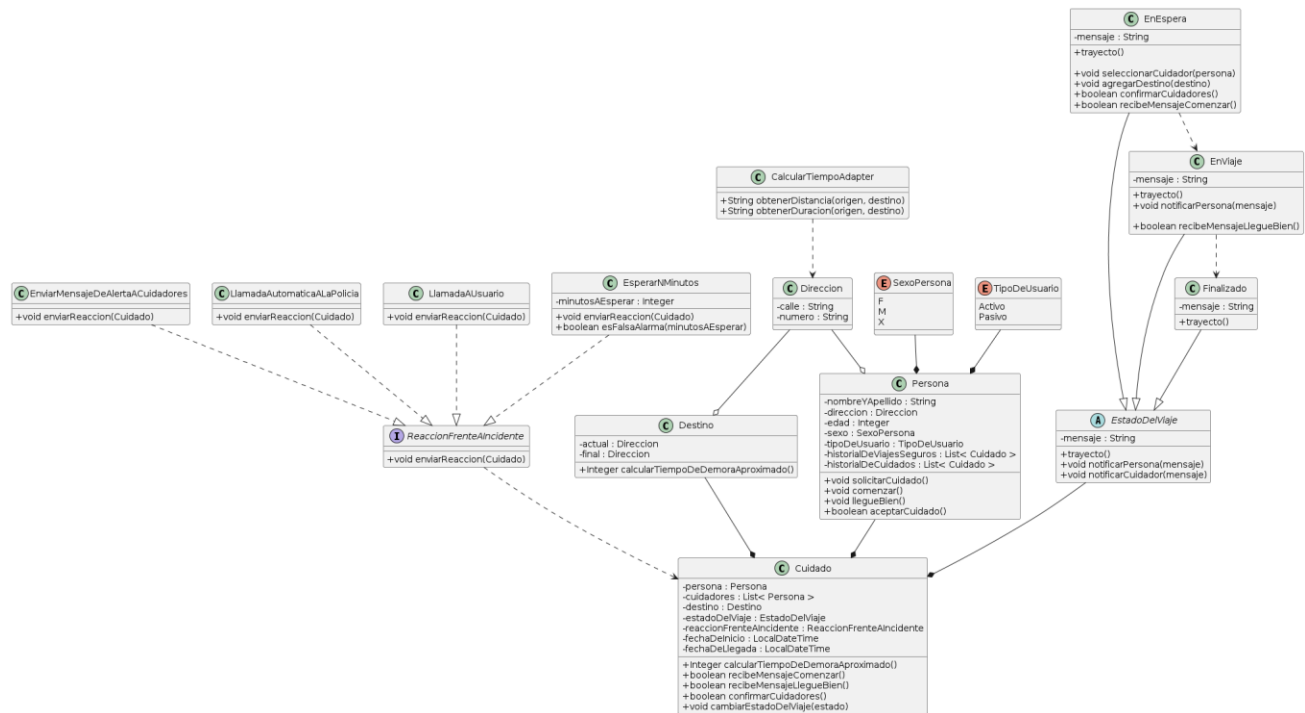
	Diseño de Sistemas K3152	Fecha <01/05/24>
	Cuidándonos	Versión <1.0>

Repositorio:

<https://github.com/sahikalu/tp-parejas-cuidandonos-Sahira-y-Brenda.git>

Punto 1:

Diagrama de Clases:



Decisiones de Diseño:


Empezamos modelando a la Persona como clase donde le incorporamos los atributos que pide en el punto 1, donde el sexo lo pusimos como un Enum; Además un tipo de usuario el cual es un Enum de tipo activo o pasivo, por el lado de la dirección pensamos en modelarla como un String pero luego nos dimos cuenta que se podía escribir de una misma manera una calle (haciendo referencia a la misma) para finalmente modelarla como clase y esta utilizarla en la clase Destino donde tenemos la dirección actual y la dirección final.

En este caso tomamos a la Persona que puede ser tanto un usuario normal como un ciudadano, en donde le agregamos un método solicitarCuidado(), comenzar(), llegueBien() y aceptarCuidado().

Decimos hacer una clase Ciudadano la cual va a contener a un transeúnte (Persona), una lista de cuidadores (de tipo Persona) y una lista de destinos de tipo Destino, como métodos agregamos calcularTiempoDeDemoraAproximado(), recibeMensajeComenzar(), recibeMensajeLlegueBien(), confirmarCuidadores() y cambiarEstadoDelViaje(estado)

Utilizamos el Patron State en la clase abstracta Estado de Viaje la cual contienen los métodos notificarPersona(mensaje), notificarCuidador(mensaje) y trayecto() en donde este último manejará el cambio de estado entre las clases hijas, así como en cada estado manejará lo que hace la clase.

Para cada momento del viaje tenemos una instancia distinta, las cuales son EnEspera, EnViaje y Finalizado, en la primera clase, se selecciona y se confirma o no el cuidador, se inserta el destino, se envía el mensaje comenzar. Al recibir el mensaje de comenzar, automáticamente se pasa al otro estado (EnViaje) (en el único en el que sobrescribimos el método notificarPersona(mensaje) para inhabilitar las notificaciones) y una vez finalizado el trayecto (es decir, habiendo recibido el mensaje finalizado) se pasa al último estado (Finalizado).

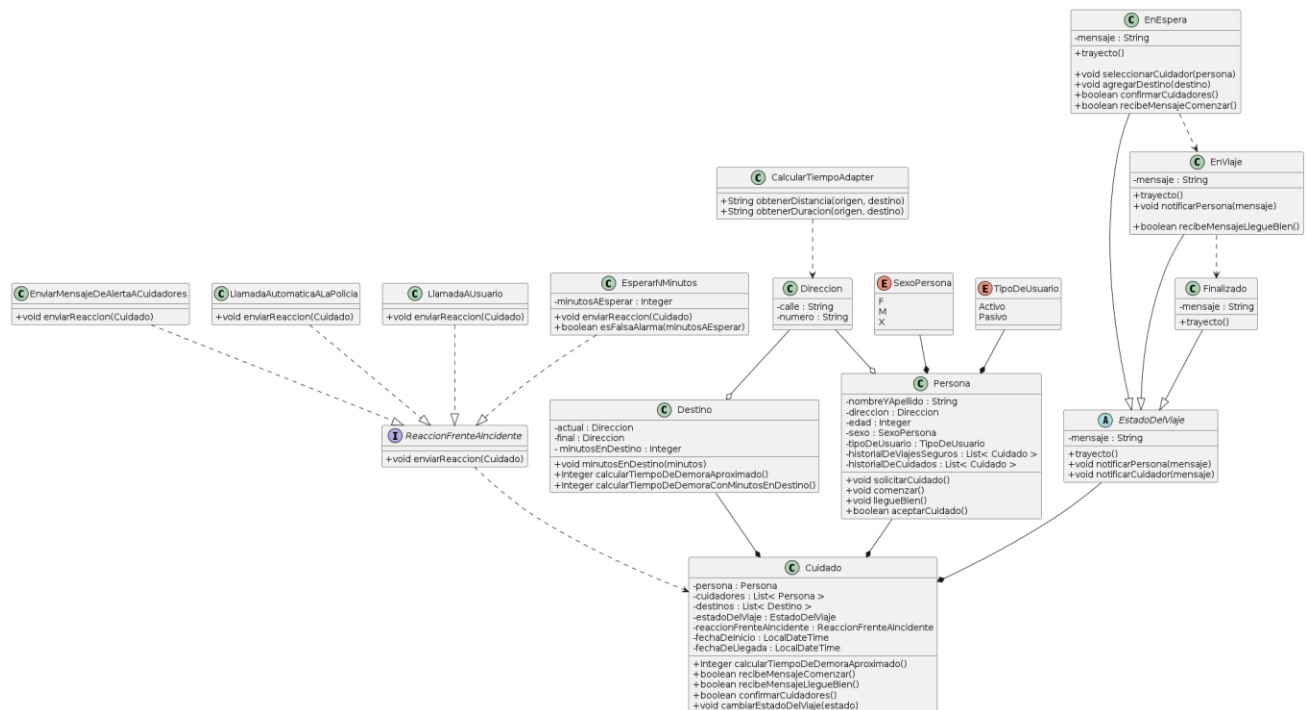
	Diseño de Sistemas K3152	Fecha <01/05/24>
	Cuidándonos	Versión <1.0>

También utilizamos el Patrón Templated Method en la parte de las reacciones, donde tenemos una interfaz `ReaccionFrenteAlIncidente` la cual tiene como método `enviarReaccion()` que recibe un cuidado y de las cuales extienden las clases `EnviarMensajeDeAlerta()`, `LlamadaAutomaticaALaPolicia()`, `LlamadaAlUsuario()`, `EsperarNMinutos()` donde esta última tiene un atributo `minutosAEsperar` y un metodo booleano `esFalsaAlarma()`, en donde cada clase completa la plantilla sobre la reacción que tiene que enviar.

Por último, utilizamos el Patrón Adapter para la unión con la API en donde le mandamos los datos que requiere y con los datos obtenidos calculamos el tiempo de demora aproximado entre origen y destino.

Punto 2:

Diagrama de Clases:



Decisiones de Diseño:

Como en este requerimiento el transeúnte puede tener más de una parada a diferencia del anterior, decimos modelar una lista de destinos. En la cual el orden de ingreso es el orden de ejecución.

Para el calculo de las demoras entre 2 destinos, hicimos el método `calcularTiempoDeDemoraAproximado` (en el Destino). Para saber la demora del inicio al final del trayecto, decidimos crear `calcularTiempoDeDemoraConMinutosEnDestino` en el que reutilizamos el método anterior y agregamos los minutos.