**Justificaciones diagrama**

1) para solicitar el viaje, ingresa un Viaje, es decir direccion final e inicial y selecciona a los cuidadores. Se envían estos datos a los cuidadores y estos deciden si aceptar o no. En caso de que ninguno acepte, se asume que no se hace el viaje. El método solicitarViaje(viaje: Viaje), envía la notificación a todos los cuidadores seleccionados, de acuerdo que seleccionen devuelve el viaje con igual cantidad o menos cuidadores.

2) Luego que se procese las solicitudes de los cuidadores y si al menos uno de los cuidadores acepta el viaje la variable puedeIniciarViaje cambiaria a True. Entonces el transeúnte decide cuando iniciar el viaje con el método: comenzarViaje(viajeFinal: Viaje), este viaje es el viaje devuelto por la función solicitarViajeA(viaje: Viaje)

3) cuando se comienza el viaje se cambia el valor de estatus cambia al valor de enViaje y de esta forma no puede recibir notificaciones, ya que puedeRecibirNotificacion(), se activa cuando esta por entrar una notificación, si enViaje = True => no puede recibir notificaciones.

3.1) en caso que algo malo suceda, se cambiaría el estatus a huboIncidente, se avisaría al sistema y este llamaría a su método actuarAnteIncidente(formaAnteIncidente: FormaAnteIncidente), de acuerdo a la forma seleccionada que tenga el usuario. Esta forma se puede cambiar a futuro.

4) por último, si el transeúnte llega bien, entonces el presionaría el botón de llegue bien, estatus cambia a llegoBien, entonces puede recibir notificaciones.

notificarViaje(cuidadores: List<Cuidador>), se encarga de avisar a todos los cuidadores del nuevo viaje y si aceptan o no. Los que no acepta se eliminan de la lista original. El cuidador recibe la notificación de la solicitud de un viaje y decide si aceptar o no.  
  
Para el diagrama las formas de actuar ante un incidente se usó el patrón de diseño: State. Se usa ya que dependiendo la forma que tenga seleccionada el usuario se actúa de una forma u otra ante un incidente. El comportamiento depende del estado o su forma. también es un State y no Strategy porque en este caso si hay una posibilidad de cambio de estados.

**Punto 2)**

Para este punto los cambios que hubieron fueron que ahora el viaje tiene una lista de la clase Parada en lugar de una única Parada. Y a esta clase, se le agregaron el método tiempoDetenidoMayorA0() y se agrega el tiempoDetenido, el cual es el aproximado que se detendría el transeúnte entre paradas.

Luego para Viaje se agregan calculoAproximadoTotal(), la cual es el pseudocódigo de abajo.

**Pseudocódigo**

pseudocodigo de demoras:

public void Integer calculoAproximadoTotal(List paradas){

Integer tiempoTotal;

Integer tiempoDemoraTotalAproximado = paradas.forEach(parada => parada.tiempoDemoraAproximado())

if(paradas.forrall({parada -> parada.tiempeDetenidoMayorA0()})){

// SUMA EN CASO DE QUE HAYA PARADAS

tiempoTotal = paradas.tiempoDetenido.sum() + tiempoDemoraTotalAproximado;

}

// CASO QUE NO SE DETIENE EN NINGUNA PARADA

else {

tiempoTotal = tiempoDemoraTotalAproximado;

}

return tiempoTotal

}