Consideraciones y Decisiones V.1

Clase Persona: Se diagramó solo una clase persona debido a que hacer dos clases distintas era tener información duplicada, dado que un mismo usuario puede cumplir los dos roles (un transeúnte puede ser cuidador en otro viaje y viceversa). La distinción de los roles se hace mediante la clase viaje donde también tiene en cuenta si es usuario activo o no (atributo de la clase Persona).

Reacción Incidente: Se utilizó un patrón strategy dado que en el enunciado especifica que hay diferentes maneras de reacción y que el usuario (Persona) elige una opción específica, además cumple los requisitos de que se pueden agregar nuevas reacciones en un futuro y que el usuario puede cambiar su preferencia. FalsaAlarma tiene en cuenta el margen de tiempo que se especificó para realizar esa reacción elegida, por eso mismo las tiene como atributos.

DistanceMatrixApi: Para utilizar la API de Google para calcular la distancia del viaje (utilizada para el cálculo de tiempo estimado), se decidió usar un patrón adapter para poder hacer el puente entre lo planteado y lo necesario para implementar la API. Se creó la interface AdapterCalculador utilizada por CalculadorDeTiempo para poder cumplir con el requerimiento de calcular el tiempo estimado.

Consideraciones y Decisiones V.2

Agregamos funcionalidades a la **clase viaje**, las cuales son:

- Atributo ubicación actual: Debido a que ahora se pueden agregar paradas, decidimos guardarnos la ubicación actual, que sería la parada en la que se encuentra en el momento del viaje.
- Atributo paradas: Agregamos una lista de paradas, en caso de que sea un viaje con un único destino se trabaja con una lista de un solo elemento, para evitar la repetición de lógica de código.
- Atributo minutosQueSeDetieneEnParada: Entendimos del enunciado que o para en todas las paradas (N minutos) o que nunca se detiene. Este atributo determina ese valor, en caso de que no pare es 0 minutos.
- Método calculoTiempoDemoraTotal y calculoTiempoParcialParadas: Utiliza el calculoTiempoDemora de Parada, sumatoria calculada en calculo TiempoParcialParadas y le suma el tiempo de minutosQueSeDetieneEnParada (si es necesario) para cálculo total.

Clase Parada: Optamos por la incorporación de la clase parada, la cual contiene el destino, su numeración (orden en la lista), el tiempo estimado parcial que tardaría en llegar a dicha parada. Esta se calcula con la ubicación actual, que en ese momento sería la ubicación de la parada anterior. Se deberá calcular y actualizar a la ubicación de la parada actual.

Pseudocódigo Cálculo de demora:

```
//Cálculo final donde suma el tiempo de espera por parada + las demoras por cada parada
Public Int calculoTiempoDemoraTotal() {
     return (
     minutosQueSeDetieneEnParada * paradas.length() +
     this.calcularTiempoParcialParadas()
     )
}

//Por cada parada calcula el tiempo apróximado y lo suma
Public Int calcularTiempoParcialParadas() {
   Int aux = 0
   for (int i = 1; i <= this.paradas.length() ; i++) {
        aux = aux + this.calculoTiempoDemora(this.ubicacionActual) //va suma las demoras
        this.ubicacionActual(this.paradas.destino()) //actualiza parada actual a la final utilizada en el cálculo
   }
   this.ubicacionActual(this.origen())
   return aux
}
```