Punto 2 - Modelo de Dominio

# **Parte 1**

## Patrones De Diseño Utilizados

### *Patrón Strategy*

En el apartado 7. de la consigna, notamos que para la configuración de la reacción:

* Existen muchas formas de realizar la misma acción (reaccionar), pero con distintos pasos, en el mismo objeto.
* Se requiere permitir al usuario configurar en el momento de ejecución la forma en que un objeto realizará una acción.

Por dicha razón, creamos una clase abstracta “EstrategiaDeReaccion” con el método reacciona que recibe a IrADestino como parámetro, y creamos las clases que heredan de dicha clase, que implementaran sus propios pasos dentro del método.

### *Patrón Adapter*

En el apartado 4. de la consigna, se nombra la utilización de “Distance Matrix API” y su interfaz REST para poder calcular la distancia entre dos direcciones. Al no poder conocer cómo está implementada, pero aún así necesitar avanzar con nuestro modelado, decidimos utilizar un Patrón adapter para resolver este problema. “CalcularTiempo” sería nuestro cliente, “CalcularDistanciaAdapter” nuestra interfaz Adaptadora y “DistanceMatrixAPI” sería la Adaptada.

# **Parte 2**

*Paradas*  
No se tuvo en cuenta la implementación de accionar en cada parada. Sólo diseñamos el cálculo de la demora.

# Código de los métodos para el cálculo de demora aproximado

*en EsperaNMinutos*

public Demora demoraPara(List<Direccion> direcciones) {

Demora demora = new Demora();

direcciones.forEach(dir1, dir2 -> demora.agregarDemora(this.construirDemoraPorSeccion(dir1, dir2)));

return demora;

}

public DemoraPorSeccion construirDemoraPorSeccion(dir1, dir2){

List<Direccion> seccion = new ArrayList<>();

seccion.add(dir1);

seccion.add(dir2);

Integer tiempoSeccion = calculadorDeTiempo.calcularTiempoEntre(dir1, dir2);

return new DemoraPorSeccion(seccion, tiempoSeccion);

}

*en AvisoOk*

public Demora demoraPara(List<Direccion> direcciones) {

Demora demora = new Demora();

Integer demoraTotal = IntStream.range(0, direcciones.size() - 1)

.map(i -> calculadorDeTiempo.calcularTiempoEntre(direcciones.get(i), direcciones.get(i + 1)))

.reduce(0, Integer::sum);

// Utilizamos reduce en lugar de sum y especificamos el tipo de dato de retorno como Integer

demora.setDemoraTotal(demoraTotal);

return demora;

}