

**Ejercicio 1: Clase Viaje**

```
public class Viaje {

    private Double precio;
    private Integer distancia;
    private Duration duracion;
    private TipoViaje tipo;
    private List<String> trayecto;

    public Viaje (Double precio, Integer distancia, Duration duracion, TipoViaje tipo,
        String origen, String destino) {
        setPrecio(precio);
        setDistancia(distancia);
        setDuracion(duracion);
        this.tipo = tipo;
        this.trayecto = new ArrayList<String>();
        trayecto.add(origen);
        trayecto.add(destino);
    }

    public Double getVelocidadMedia() {
        Double duracion = getDuracion().getSeconds() / 3600.0;
        return getDistancia()/duracion;
    }

    public Integer getNumeroParadas() {
        return getTrayecto().size() - 2 - getNumeroTransbordos();
    }

    public String getOrigen() {
        return getTrayecto().get(0);
    }

    public String getDestino() {
        return getTrayecto().get(getTrayecto().size() - 1);
    }

    public List<String> getParadas() {
        return getTrayecto().subList(1, getNumeroParadas() + 1);
    }

    public Integer getNumeroTransbordos() {
        Integer numTransbordos = 0;
        if (getTipo().equals(TipoViaje.TRANSBORDO)) {
            numTransbordos = calcularRepetidosConsecutivos(getTrayecto());
        }
        return numTransbordos;
    }

    public Duration getDuracion() {
        return duracion;
    }

    public void setDuracion(Duration duracion) {
        Checkers.check("Duración incorrecta", duracion.compareTo(Duration.ZERO) > 0);
        this.duracion = duracion;
    }

}
```

```

private Integer calcularRepetidosConsecutivos(List<String> trayecto) {
    Integer cont = 0;
    for (int i=0; i < trayecto.size()-2; i++) {
        if (trayecto.get(i).equals(trayecto.get(i+1))) {
            cont++;
        }
    }

    return cont;
}

```

## Ejercicio 2: Factoría

```

private static Viaje parsearViaje(String linea) {
    System.out.println("Parseando...." + linea);
    Checkers.checkNotNull(linea);
    String [] trozos = linea.split(";");
    Checkers.check("Formato no válido->" + trozos.length, trozos.length == 5);
    Double precio = Double.parseDouble(trozos[0].trim());
    Integer distancia = Integer.parseInt(trozos[1].trim());
    Duration duracion = parsearDuracion(trozos[2].trim());
    TipoViaje tipo = TipoViaje.valueOf(trozos[3].trim().toUpperCase());
    List<String> trayecto = parsearTrayecto(trozos[4].trim());
    return new Viaje(precio, distancia, duracion, tipo, trayecto);
}

private static Duration parsearDuracion(String strDuracion) {
    Checkers.checkNotNull(strDuracion);
    String [] trozos = strDuracion.split(":");
    Checkers.check("Formato no válido", trozos.length == 2);
    Integer horas = Integer.parseInt(trozos[0].trim());
    Integer minutos = Integer.parseInt(trozos[1].trim());
    return Duration.ofHours(horas).plusMinutes(minutos);
}

private static List<String> parsearTrayecto(String strTrayecto) {
    String limpia = strTrayecto.replace("[", "").replace("]", "");
    String [] trozos = limpia.split(",");
    Checkers.check("Formato no válido->" + trozos.length + "-->"
        + Arrays.toString(trozos), trozos.length >= 2);
    List<String> res = Arrays.stream(trozos)
        .map(trozo -> trozo.trim())
        .collect(Collectors.toList());
    return res;
}

```

## Ejercicio 3: Tipo Contenedor

### a) hayViajesCirculares

```

public Boolean hayViajesCirculares(){
    return viajes.stream()
        .anyMatch(viaje -> viaje.getOrigen().equals(viaje.getDestino()));
}

```

**b) getMaximaDuracion**

```
public Duration getMaximaDuracion() {
    Duration res = Duration.ZERO;
    Viaje v= viajes.stream()
        .max(Comparator.comparing(Viaje::getNumeroParadas))
        .orElse(null);
    if (v != null) {
        res = v.getDuracion();
    }
    return res;
}
```

**c) añadirTiempoDescanso**

```
public void añadirTiempoDescanso (String parada, Integer minutos) {
    viajes.stream()
        .filter(viaje -> viaje.getTrayecto().contains(parada))
        .forEach(viaje -> viaje.setDuracion(
            viaje.getDuracion().plusMinutes(minutos)));
}
```

**d) getParadas**

```
public SortedSet<String> getParadas() {
    return viajes.stream()
        .flatMap(viaje -> viaje.getParadas().stream())
        .collect(Collectors.toCollection(TreeSet::new));
}
```

**e) viajeDuracionMinimaPorDestinoPorParadas**

```
public Map<String, Viaje> viajeDuracionMinimaPorDestinoPorParadas () {
    return viajes.stream()
        .collect(Collectors.groupingBy(
            v -> v.getDestino(),
            Collectors.collectingAndThen(
                Collectors.minBy(
                    Comparator.comparing(Viaje::getDuracion)),
                v -> v.get()
            )
        ));
}
```