

## Entender que hace es entender quien lo hace

en ese método (TREN APTO) te pide el destino entonces primero buscas por eso, después por la cantidad de vagones , tenes que entrar al tren después al vagón y ver si es de pasajeros, entras al primero y es de pasajeros , entonces pasajeros++ , y después vuelves y preguntas si es de carga haces carga ++ entonces cuando terminas de ver todo el tren decis los parámetros que me pasaron son iguales o mayores a los que recibí ? si es así devolves el tren si no vas al siguiente, al final pregunta si tiene maquinista y si no tiene le asignas un maquinista al tren y un tren al maquinista

este metodo devuelve un tren ya creado, entonces tenes que usar el constructor de tren supongo, y ademas si creas el tren tambien tenes que borrar el conductor , y los vagones que usaste de las listas de "vagones libres" y de " conductores libres" no se llaman asi pero se entiende

8:52 n.m

o sea que tendrias que hacer metodos de asignar conductor asignar vagones

8:53

y que ellos los devuelvan y los borren de cada array

8:53 n.r

```
public Persona buscarPorDni (+ int dniBuscado )  
{  
    + int pos  + Persona personaEncontrada ← null  
  
    /* Primero buscamos a la persona sin "caernos" de la estructura */  
  
    pos ← 0  
  
    pos < personas.size() && personas.get(pos).getDni() != dniBuscado  
  
        pos ← pos + 1  
  
    /* Solamente si no nos pasamos del último elemento... */  
  
    pos < personas.size()  
V  
    personaEncontrada ← personas.get(pos)  
F  
    return personaEncontrada
```

```
class Mesa :  
    public Jugador buscarJugador (‐ String nombre)  
    ‐int i ← 0  ‐Jugador jEncontrado ← null
```

```
        i < this.jugadores.size() && encontrado == null
```

V

```
            this.jugadores.get(i).getNombre() == nombre
```

F

```
                jEncontrado ← this.jugadores.get(i)
```

```
i++
```

```
        return jEncontrado
```

```
class Juego :  
    public Jugador buscarJugador (‐ String nombre)  
    ‐Jugador jEncontrado ← null  ‐int i ← 0
```

```
        i < this.mesas.size() && encontrado == null
```

```
            Mesa m ← this.mesas.get(i)
```

```
            jEncontrado ← m.buscarJugador(nombre)
```

```
i++
```

```
return jEncontrado
```

```
class ORTFlix:  
    public void depurarListaNegra (int monto)
```

```
    int pos ← 0
```

```
    pos < deudores.size()
```

```
    deudores.get(pos).getSaldo() ≤ monto
```

```
    deudores.remove(pos)
```

```
    pos ← pos + 1
```

```
public Persona buscarPorDni (int dniBuscado)
```

```
int pos PersonaEncontrada ← null
```

```
/* Primero buscamos a la persona sin "caernos" de la estructura */
```

```
pos ← 0
```

```
pos < personas.size() && personaEncontrada == null
```

```
    personas.get(pos).getDni() != dniBuscado
```

```
    pos ← pos + 1
```

```
    personaEncontrada ← personas.get(pos)
```

```
return personaEncontrada
```

```
class RedSocial:  
    public RedSocial (+ String nombre)  
        this.nombre = nombre  
        this.usuarios = new ArrayList<Usuario>()
```

```
class Usuario:  
    public boolean agregarAmigo (+ String nickname)  
    +boolean pudeAregar = false +Usuario usuario +Usuario amigoExistente = null
```

```
        amigoExistente = buscarAmigo(nickname)
```

```
V           amigoExistente == null F
```

```
        usuario = this.redSocial.buscarUsuario(nickname)
```

```
V           usuario != null F
```

```
        agregarAmigo(usuario)
```

```
        usuario.agregarAmigo(this.getNickname())
```

```
        pudeAregar = true
```

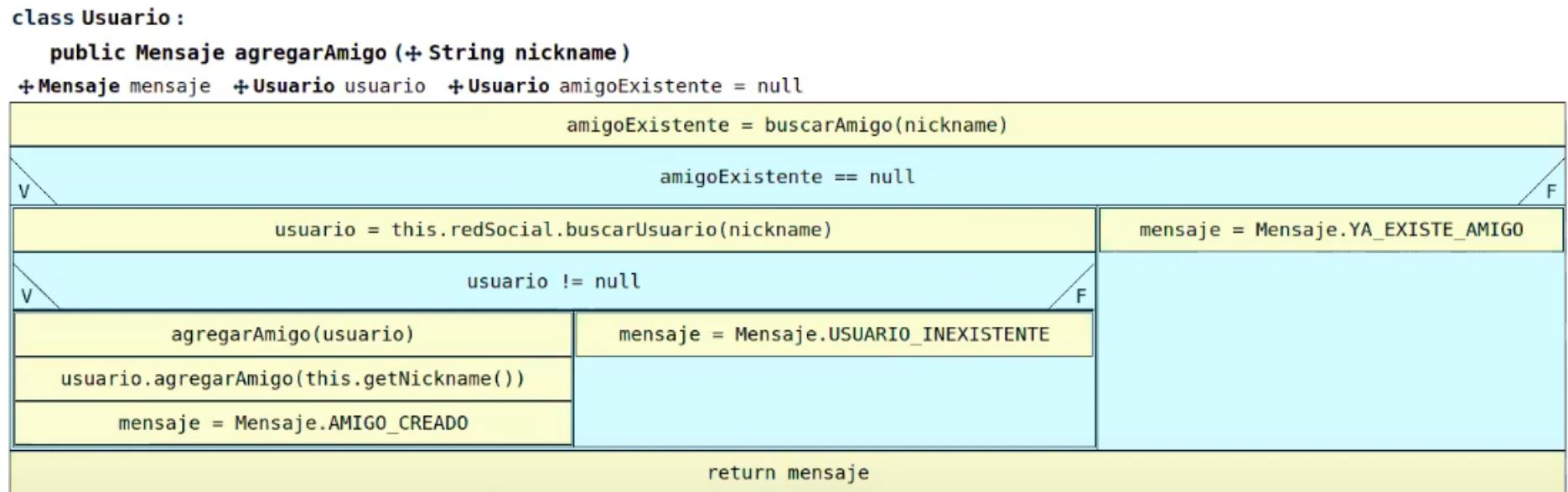
```
    return pudeAregar
```

```
class Usuario:  
    public Usuario (+ String nickname , + String nombre)
```

```
        this.nickname = nickname
```

```
        this.nombre = nombre
```

```
        this.amigos = new ArrayList<Usuario>
```



```

class RedSocial:
    public ArrayList<Resultado> obtenerUsuariosPais ()
    + int contUsuario + ArrayList<Resultado> salida + String ultimoPais=null + Resultado resultado
        resultado = new ArrayList<Resultado>()
        contUsuario = 0
    V                                     ultimoPais!=null
    V                                     ultimoPais!=usuario.getPais()          F
    resultado = new Resultado(ultimoPais, contUsuario)
    salida.add(resultado)
    contUsuario = 0
    ultimoPais = usuario.getPais()
    contUsuario++
F
return salida

```

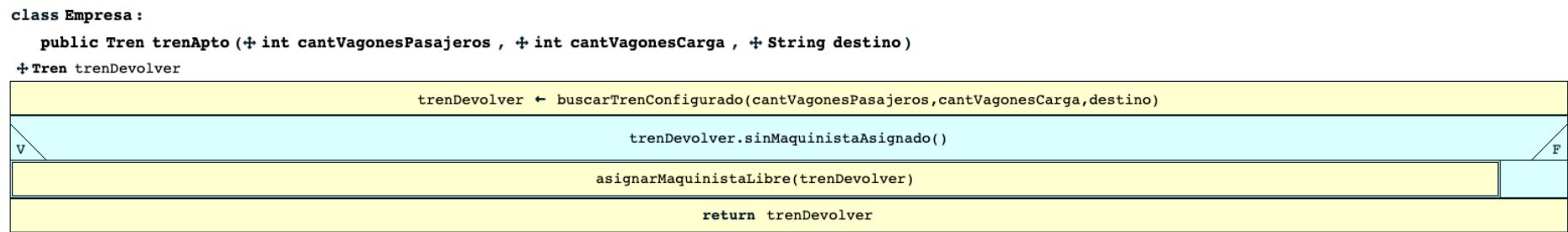
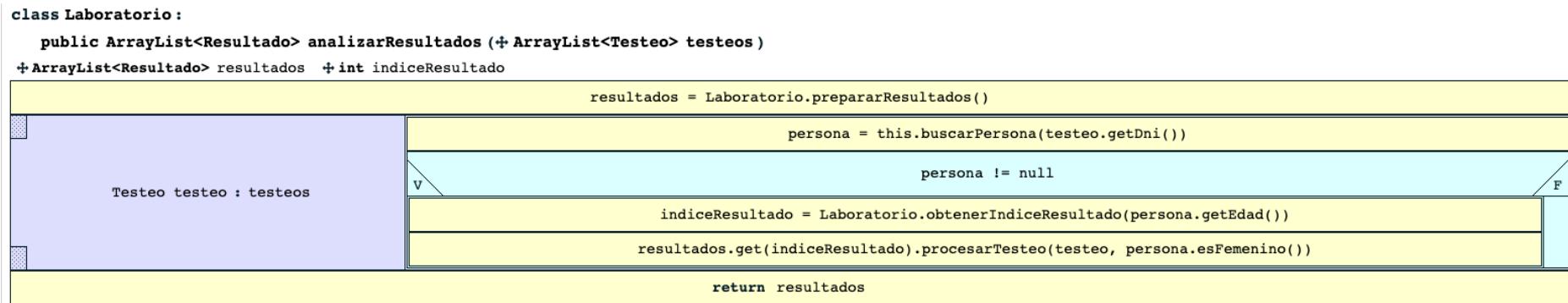
Diagram notes: The code is annotated with regions and markers. A purple shaded region on the left contains the variable declaration 'Usuario usuario : this.usuarios'. The main code area has several colored regions: yellow for the initial assignment of 'resultado' and 'contUsuario', light blue for the condition 'ultimoPais!=null', orange for the nested condition 'ultimoPais!=usuario.getPais()', and light green for the final increment of 'contUsuario'. A vertical bar on the far left has small purple squares at its top and bottom. The word 'F' appears twice in the diagram, once above the nested condition and once below the final increment of 'contUsuario'.

```

class Laboratorio:
    private static ArrayList<Resultado> prepararResultados ()
    + final tipo CANT_RANGOS ← 4 + ArrayList<Resultado> resultados ← new ArrayList<Resultado>()
    int cont ← 1 , CANT_RANGOS , 1
    resultados.add(new Resultado())
    return resultados

```

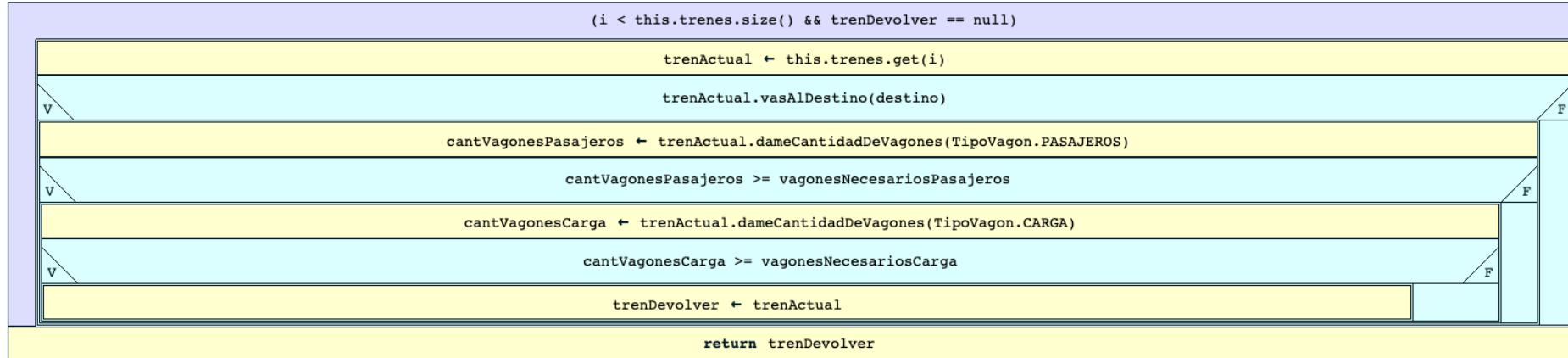
Diagram notes: This code is shown in a single row with a light blue background. It includes a small purple square icon on the far left and a yellow square icon on the right. The variable 'CANT\_RANGOS' is annotated with a plus sign and the value '4'. The variable 'resultados' is annotated with a plus sign and the type 'ArrayList<Resultado>'. The word 'return' is also annotated with a plus sign and the variable 'resultados'.



```

class Empresa:
    private Tren buscarTrenConfigurado ( + int vagonesNecesariosPasajeros , + int vagonesNecesariosCarga , + String destino)
    +Tren trenDevolver = null + int i ← 0 +Tren trenActual + int cantVagonesCarga + int cantVagonesPasajeros

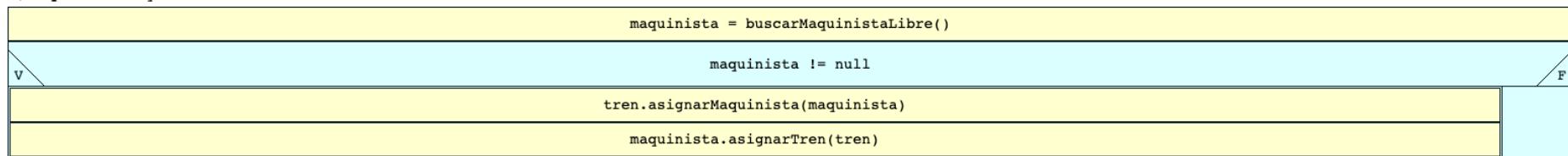
```



```

class Empresa:
    private void asignarMaquinistaLibre ( + Tren tren )
    +Maquinista maquinista

```



```

class Recetario:
    public ArrayList<IngredienteAReponer> prepararListadoAComprar (
        +String nombreReceta, double porciones, Heladera heladera, Alacena alacena)
    +Receta recetaBuscada +ArrayList<IngredienteAReponer> listaCompras
        receta ← buscarReceta(nombreReceta)
    V
        receta != null
    F
        listaCompras ←
            compararIngredientes(receta.dameIngredientes(), porciones, heladera, alacena)
    S "Receta no encontrada"
        listaCompras ←
            new ArrayList<IngredienteAReponer>
    return listaCompras

```

```

class Receta:
    public ArrayList<Ingrediente> dameIngredientes ()
        return this.ingredientes

```

```

class Recetario:
    public ArrayList<IngredienteAReponer> compararIngredientes (
        +ArrayList<Ingrediente> ingredientes, +double porciones, +Heladera heladera, +Alacena alacena)
    +ArrayList<IngredienteAReponer> listaFaltantes +double cuantoNecesito
        listaFaltantes ← new ArrayList<IngredienteAReponer>()
    V
        Ingrediente ingrediente :
            ingredientes
                ingrediente.esRefrigerado()
    F
        cuantoNecesito ← alacena.cuantoNecesita(ingrediente.getNombre(), porciones)
    V
        cuantoNecesito > 0
    F
        listaFaltantes.add(new IngredienteAReponer(ingrediente.getNombre(), cuantoNecesito ))
    return listaFaltantes

```

```
public Carrera buscarCarrera (‐ String fecha)
+ int cantCarreras + Carrera carreraActual + int pos ← 0 + Carrera carreraDevolver ← null
    cantCarreras ← this.carreras.size()
    (carreraDevolver == null) y (pos < cantCarreras)
        carreraActual ← this.carreras.get(pos)
        V (carreraActual.getFecha() == fecha) F
            carreraDevolver ← carreraActual
            pos++
    return expresión

public int dameCantVueltas ()
    return this.vueltas.size()

public String toString ()
    return "Nombre: " + this.nombre + " - " + "DNI: " + this.dni
```

```
public double calcularPromedio ()  
    + double sumaTiempos ← 0    + int cantVueltas    + double promDevolver
```

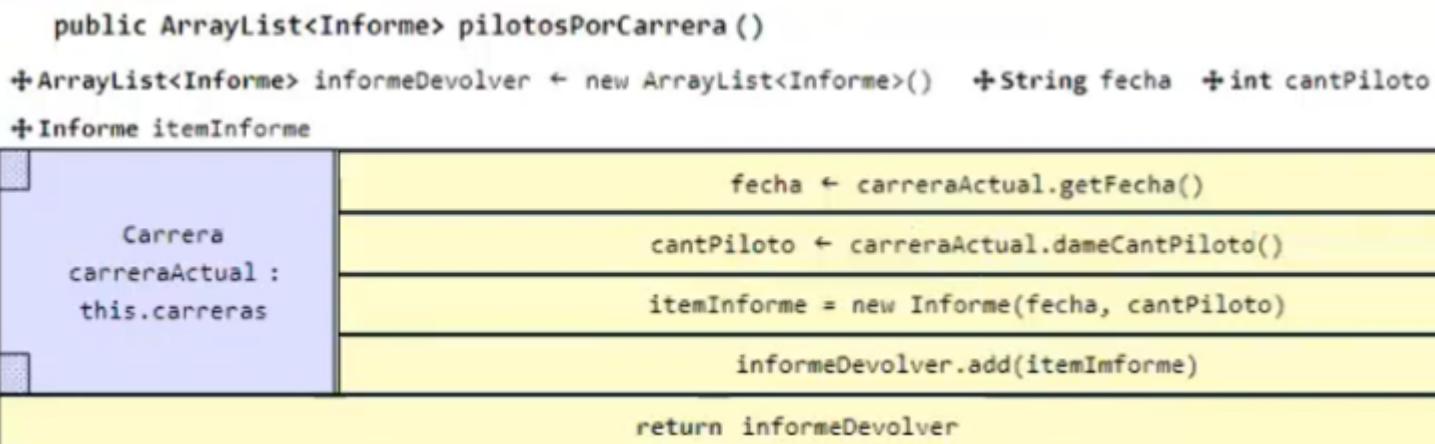
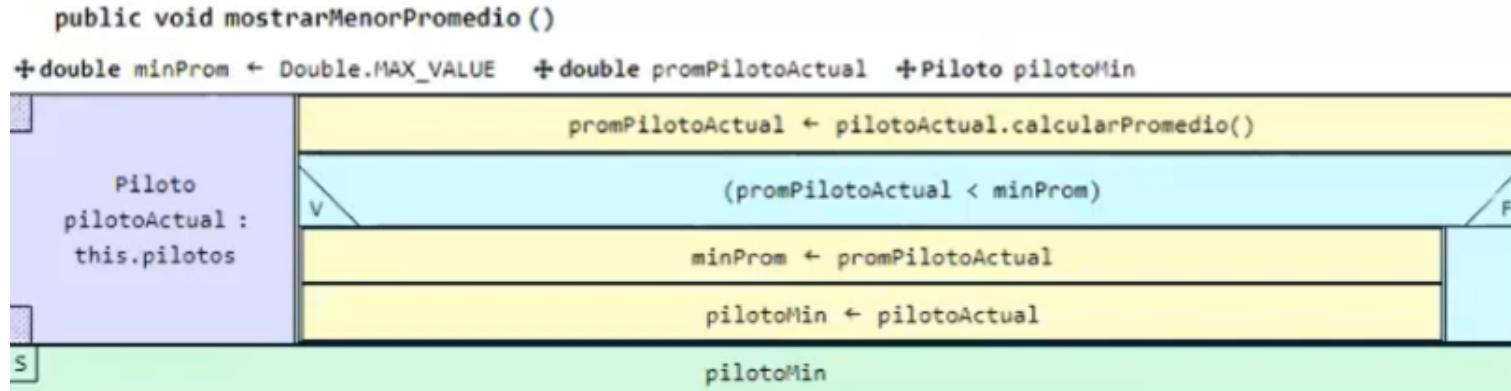
Vuelta vueltaActual : this.vueltas	sumaTiempos ← sumaTiempos + vueltaActual.getTiempo()
	cantVueltas ← this.vueltas.size()
	promDevolver ← Matematica.obtenerPromedio(sumaTiempos, cantVueltas)
	return <u>promDevolver</u>

```
public ArrayList<Piloto> buscarPilotosPorDeabajoDe (+ double tiempo)  
+ final int VUELTAS_MIN ← 10    + ArrayList<Piloto> pilotosDevolver ← new ArrayList<Piloto>()
```

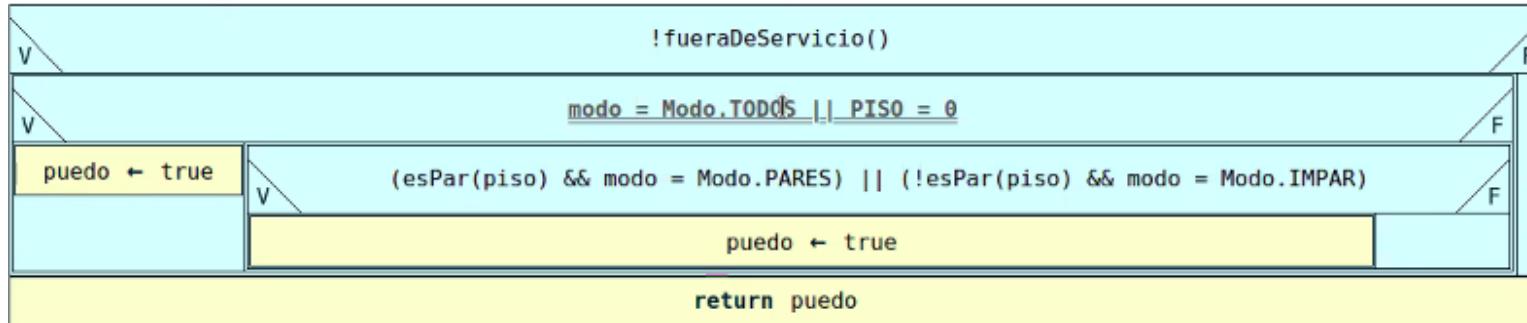
Piloto pilotoActual : this.pilotos	(pilotoActual.dameCantidadVueltas() < VUELTAS_MIN) y (pilotoActual.calcularPromedio() ≤ tiempo) pilotosDevolver.add(pilotoActual)
	return pilotosDevolver

```
class Reparadora :  
    public Reparadora ()  
        autobuses ← new ArrayList<Autobus>()
```

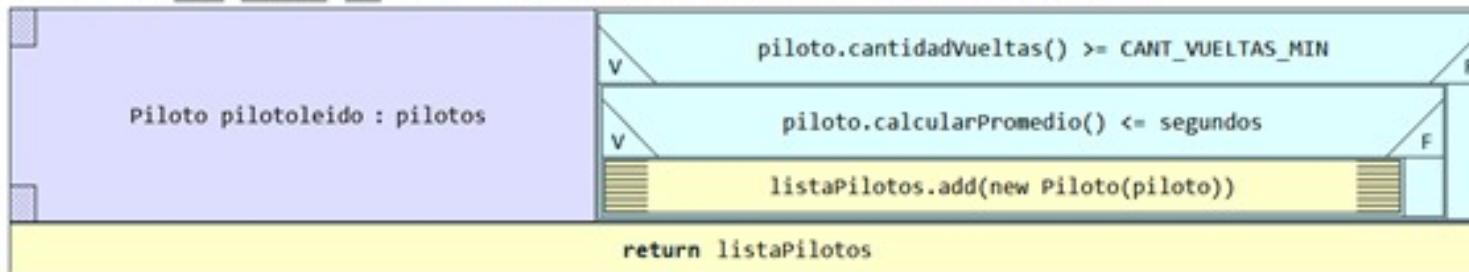
```
class Informe :  
    public Informe (+ String fechaCarrera , + int cantPiloto)  
        this.fechaCarrera ← fechaCarrera  
        this.cantPiloto ← cantPiloto
```



```
class Ascensor :  
    public boolean puedoParar (+ int piso)  
    +boolean puedo ← false
```



```
class Carrera :  
    public ArrayList<Piloto> buscarPilotosPorDeabajoDe (+ double segundos)  
    +final int CANT_VUELTAS_MIN ← 10 +ArrayList<Piloto> listaPilotos ← new ArrayList<>
```



---

AUTOBUS

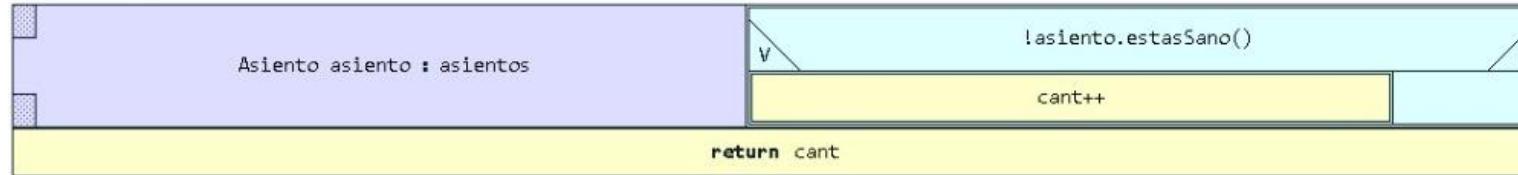
```
class Autobus :  
    private void colocarAsientos (‐ ArrayList<Asiento> asientosOK)  
        asientosOK.size() > 0  
        asientos.add(asientosOK.remove(0))
```

```
class Autobus :  
    public int dameCantidadFaltantes ()  
        return cantAsientos - asientos.size()
```

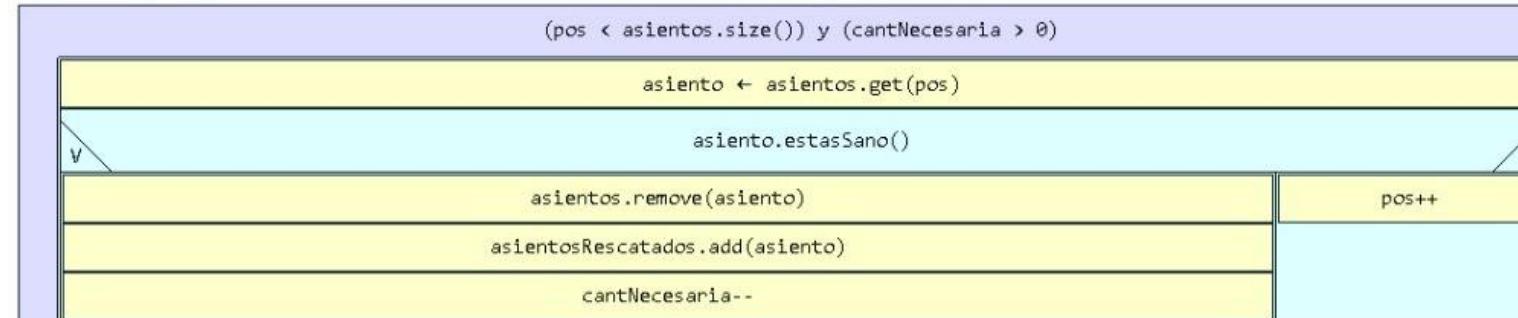
```
class Asiento :  
    public boolean estasSano ()  
        return (this.estado==Estado.SANO)
```

```
class Autobus :  
    public boolean estasCompletoBuenEstado ()  
        ‐boolean resultado ← true  ‐int pos ← 0  ‐Asiento asiento  
            pos < asientos.size() && resultado  
            asiento ← asientos.get(pos)  
            resultado ← asiento.estasSano()  
            pos++  
        return resultado
```

```
class Autobus :  
    public int dameCantidadAsientosMalos ()  
    {int cant ← 0
```



```
class Autobus :  
    public void dameAsientosBuenos (int cantNecesaria , ArrayList<Asiento> asientosRescatados )  
    {int pos ← 0 Asiento asiento
```



```

class Autobus :
    public ArrayList<Asiento> quitarAsientosMalEstado()
        +ArrayList<Asiento> asientosMalos + Asiento asiento
            asientosMalos ← new ArrayList<Asiento>()
            pos < asientos.size()
            asiento ← asientos.get(pos)
            !(asiento.estasSano())
            V
            asientos.remove(asiento) pos++
            asientosMalos.add(asiento)
            return asientosMalos

```

```

class Autobus :
    private void colocarAsientos (+ ArrayList<Asiento> asientosOK)
        asientosOK.size() > 0
        asientos.add(asientosOK.remove(0))

```

```

class Reparadora :
    public ArrayList<Asiento> completarAutobus (+ Autobus autobus )
        +ArrayList<Asiento> asientosMalos + int cantNecesaria + ArrayList<Asiento> asientosOK
            autobuses.remove(autobus)
            asientosMalos = autobus.quitarAsientosMalEstado()
            cantNecesaria ← autobus.dameCantidadFaltantes()
            asientosOK = rescatarAsientosBuenos(cantNecesaria)
            autobus.colocarAsientos(asientosOK)
            autobuses.add(autobus)
            return asientosMalos

```

```

class Reparadora:
    public ArrayList<Asiento> completarAutobus ( Autobus autobus )
    +ArrayList<Asiento> asientosMalos  +int cantNecesaria +ArrayList<Asiento> asientosOK
        autobusesRotos.remove(autobus)
        asientosMalos = autobus.quitarAsientosMalEstado()
        cantNecesaria ← autobus.dameCantidadFaltantes()
        asientosOK = rescatarAsientosBuenos(cantNecesaria)
        autobus.colocarAsientos(asientosOK)
    V                                     F
        autobus.estasCompletoBuenEstado()
    autobusesVenta.add(autobus)           autobusesRotos.add(autobus)
    return asientosMalos

```

```

class Reparadora:
    private ArrayList<Informe>() obtenerInforme()
    +Informe informe +ArrayList<Informe>() listaInforme
        listaInforme ← new ArrayList<Informe>()
        Autobus autobus : autobuses
        informe = new Informe(autobus.getPatente(),autobus.dameCantidadAsientosMalos())
        listaInforme.add(informe)
    return listaInforme

```

```

class Reparadora:
    public void mostrarInforme()
    +ArrayList<Informe>() listaInforme
        listaInforme = obtenerInforme()
        Informe informe :
            Patente: " + informe.getPatente() + " - " + "Asientos Malos: " + informe.getCantAsientosMalos()

```

```

class Reparadora:
    private ArrayList<Asiento> rescatarAsientosBuenos (⊕ int cantNecesaria)
    +ArrayList<Asiento> asientosObtenidos  ⊕ int pos ← 0
        asientosObtenidos ← new ArrayList<Asiento>()
        pos < autobuses.size() && cantNecesaria > 0
            autobusActual ← autobuses.get(pos)
            V
                !autobusActual.estasCompletoBuenEstado()
            F
                autobusActual.dameAsientosBuenos(cantNecesaria, asientosObtenidos)
                cantNecesaria ← cantNecesaria - asientosObtenidos.size()
                pos++
        return asientosObtenidos

```

```

class Informe:
    public Informe(⊕ String patente , ⊕ int cantAsientosMalos )
        this.patente ← patente
        this.cantAsientosMalos ← cantAsientosMalos

```

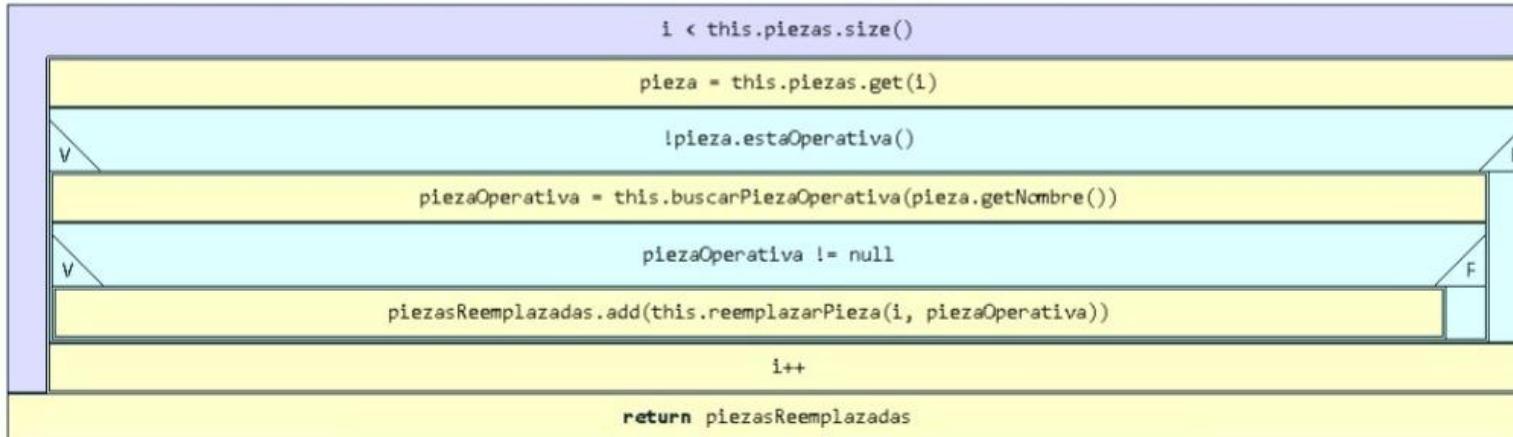
---

## DROIDES

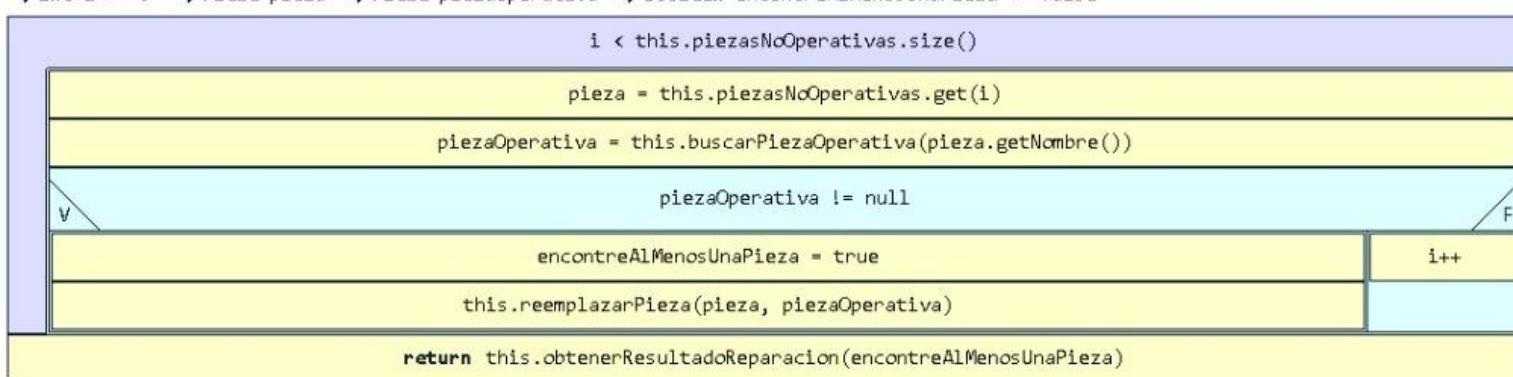
```
class Droide:  
    public Pieza buscarPiezaOperativa(⊕ String nombrePieza)  
    {  
        int i ← 0    +Droide droide    +Pieza piezaEncontrada  
  
            i < droidesFueraDeServicio.size() && piezaEncontrada == null  
                droide = droidesFueraDeServicio.get(i)  
                piezaEncontrada = droide.obtenerPiezaOperativa(nombrePieza)  
                i++  
  
        return piezaEncontrada  
    }
```

```
class Droide:  
    private Pieza obtenerPiezaOperativa(⊕ String nombrePieza)  
    {  
        int i ← 0    +Pieza pieza    +Pieza piezaEncontrada  
  
            i < this.piezas.size() && piezaEncontrada == null  
                pieza = this.piezas.get(i)  
                pieza.getNombre().equals(nombrePieza) && pieza.estaOperativa()  
                    piezaEncontrada = this.piezas.remove(i)    F  
                    i++  
  
    return piezaEncontrada  
    }
```

```
class Droide:  
    private ArrayList<Pieza> autoRepararse()  
        +int i ← 0  +Pieza pieza  +Pieza piezaOperativa  +ArrayList<Pieza> piezasReemplazadas ← new ArrayList<Pieza>()
```



```
class Droide:  
    private ResultadoReparacion autoRepararse()  
        +int i ← 0  +Pieza pieza  +Pieza piezaOperativa  +boolean encontreAlMenosUnaPieza ← false
```



```

class Droide:
    private void reemplazarPieza(♦ Pieza piezaNoOperativa , ♦ Pieza piezaOperativa)
        this.piezasNoOperativas.remove(piezaNoOperativa)
        this.piezasOperativas.add(piezaOperativa)

```

```

class Droide:
    private ResultadoReparacion obtenerResultadoReparacion (♦ boolean encontreAlMenosUnaPieza)
    ♦ ResultadoReparacion resultado ← ResultadoReparacion.REPARACION_IMPOSIBLE
    V
        this.piezasNoOperativas.isEmpty()
    F
        resultado = ResultadoReparacion.COMPLETAMENTE_OPERATIVO
        V
            encontreAlMenosUnaPieza
            F
                resultado = ResultadoReparacion.REPARACION_PARCIAL
        F
    F
    return resultado

```

```

class Droide:
    private Pieza reemplazarPieza(♦ int indice , ♦ Pieza piezaOperativa)
    ♦ Pieza piezaNoOperativa
        /* Opcion 1: con add con indice */
        piezaNoOperativa = this.piezas.remove(indice)
        this.piezas.add(indice, piezaOperativa)
        return piezaNoOperativa
        /* Opcion 2: con set */
        return this.piezas.set(i, piezaOperativa)

```

```
public boolean actualizarDomicilio (+ String domicilio)
+boolean modificado = false
V
    domicilio!=""
    setDomicilio(domicilio)
    modificado = true
return modificado
```

```
class Agenda :
    public boolean agregarPersona (+ int dni , + String nombre , + String apellido , + String domicilio)
    +boolean agregada = false +Persona personaNueva
V
    buscarPersona(dni) == null
    personaNueva = new Persona(dni, nombre, apellido, domicilio)
    this.personas.add(personaNueva)
    agregada = true
return agregada
F
```

---

## CAMION

```
class Empresa :
    private void agregarPendiente (+ Paquete paquete , + Motivo motivo)
        registrosPendientes.add(new RegistroPendiente(paquete,motivo))
```

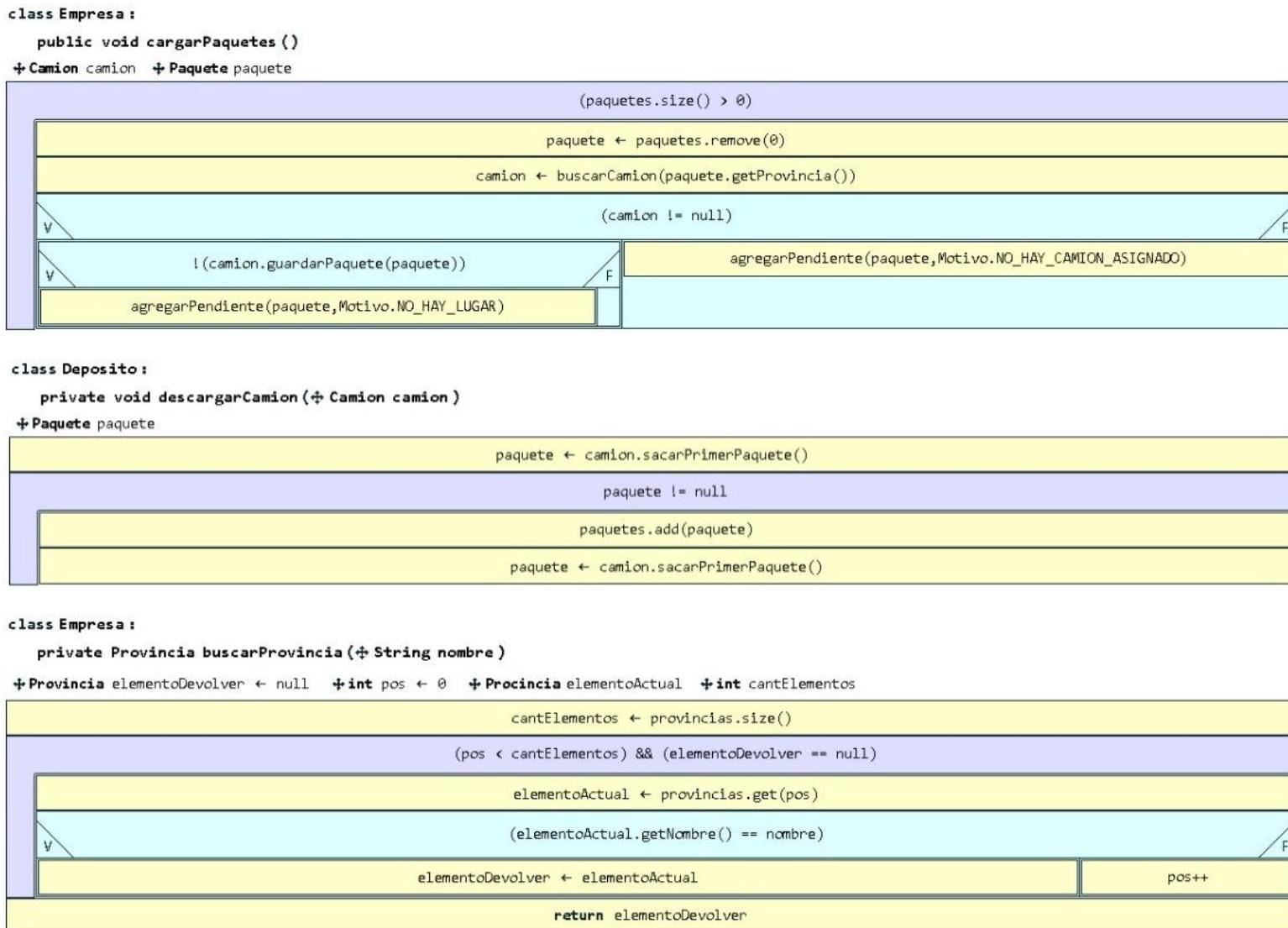
```

class Camion :
    public double getPorcentajeDeCarga()
        return pesoActualCarga * 100 / TOPE_PESO

class Empresa :
    public boolean agregarPaquete(† double peso , † double valor , † String nombreProvincia)
    †boolean resultado ← false
        provincia ← buscarProvincia(nombreProvincia)
        (provincia != null)
        paquetes.add(new Paquete(peso,valor,provincia))
        resultado ← true
    return resultado

class Empresa :
    private Camion buscarCamion(† Provincia provincia)
    †Camion elementoDevolver ← null  †int pos ← 0  †Camion elementoActual  †int cantElementos
        cantElementos ← camiones.size()
        (pos < cantElementos) && (elementoDevolver == null)
        elementoActual ← camiones.get(pos)
        (elementoActual.getProvincia() == provincia)
        elementoDevolver ← elementoActual
        pos++
    return elementoDevolver

```



```
class Camion :  
    public Camion(† String patente )  
        setPatente(patente)  
        pesoActualCarga ← 0  
        paquetes ← new ArrayList<>()
```

```
class EmpresaDeTransporte :  
    public EmpresaDeTransporte(† String nombre )  
        setNombre(nombre)  
        depositos ← new ArrayList<>()
```

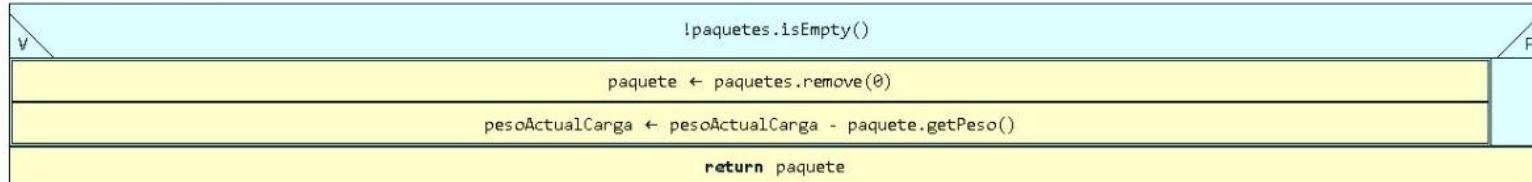
```
class Camion :  
    public boolean guardarPaquete († Paquete paquete )  
    +boolean resultado  
        resultado ← hayCapacidad(paquete.getPeso())  
        resultado  
        pesoCargaActual ← pesoCargaActual + paquete.getPeso()  
        carga.add(paquete)  
        return resultado
```

```
class Camion :  
    private boolean hayCapacidad(† double peso )  
        return (pesoCargaActual + peso <= cargaMaxima)
```

```

class Camion :
    public Paquete sacarPrimerPaquete()
    +Paquete paquete ← null

```



```

class EmpresaDeTransporte :
    private ArrayList<String> obtenerDepositosSegunCriteriao(⊕ int cantidadDeCamiones , ⊕ double porcentajeMinimo)
    +ArrayList<String> provincias

```



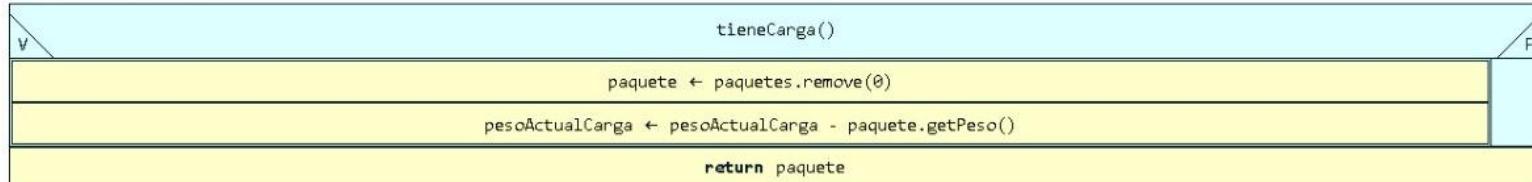
```

class Deposito :
    private RESULTADO_DISTRIBUCION quitarCamionPorDesperfecto(⊕ String patente )
    +Paquete paquete +Camion camion +ResultadoDistribucion resultado +boolean pudo ← true

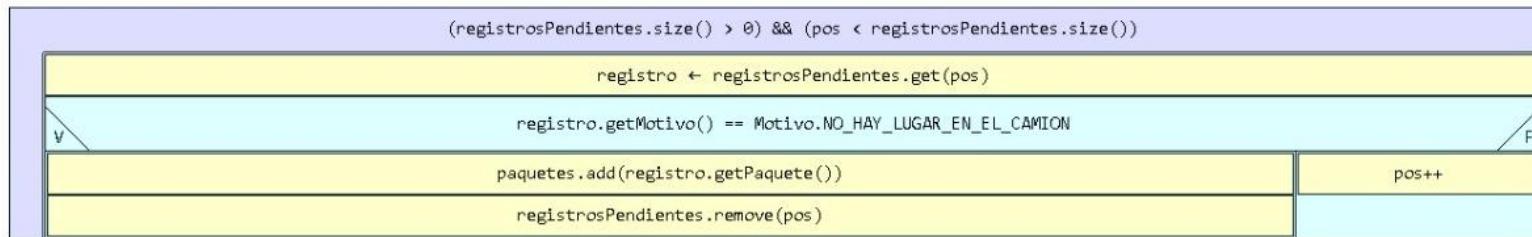
```



```
class Camion :  
    public Paquete sacarPrimerPaquete()  
    {  
        Paquete paquete = null
```

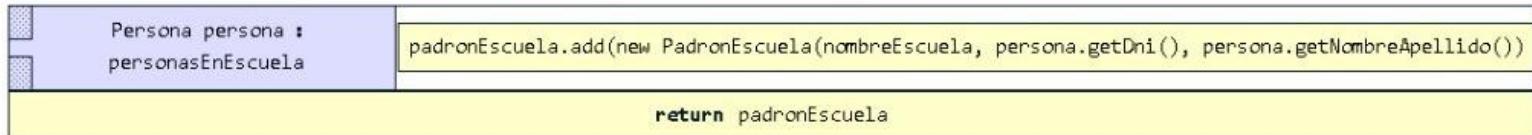


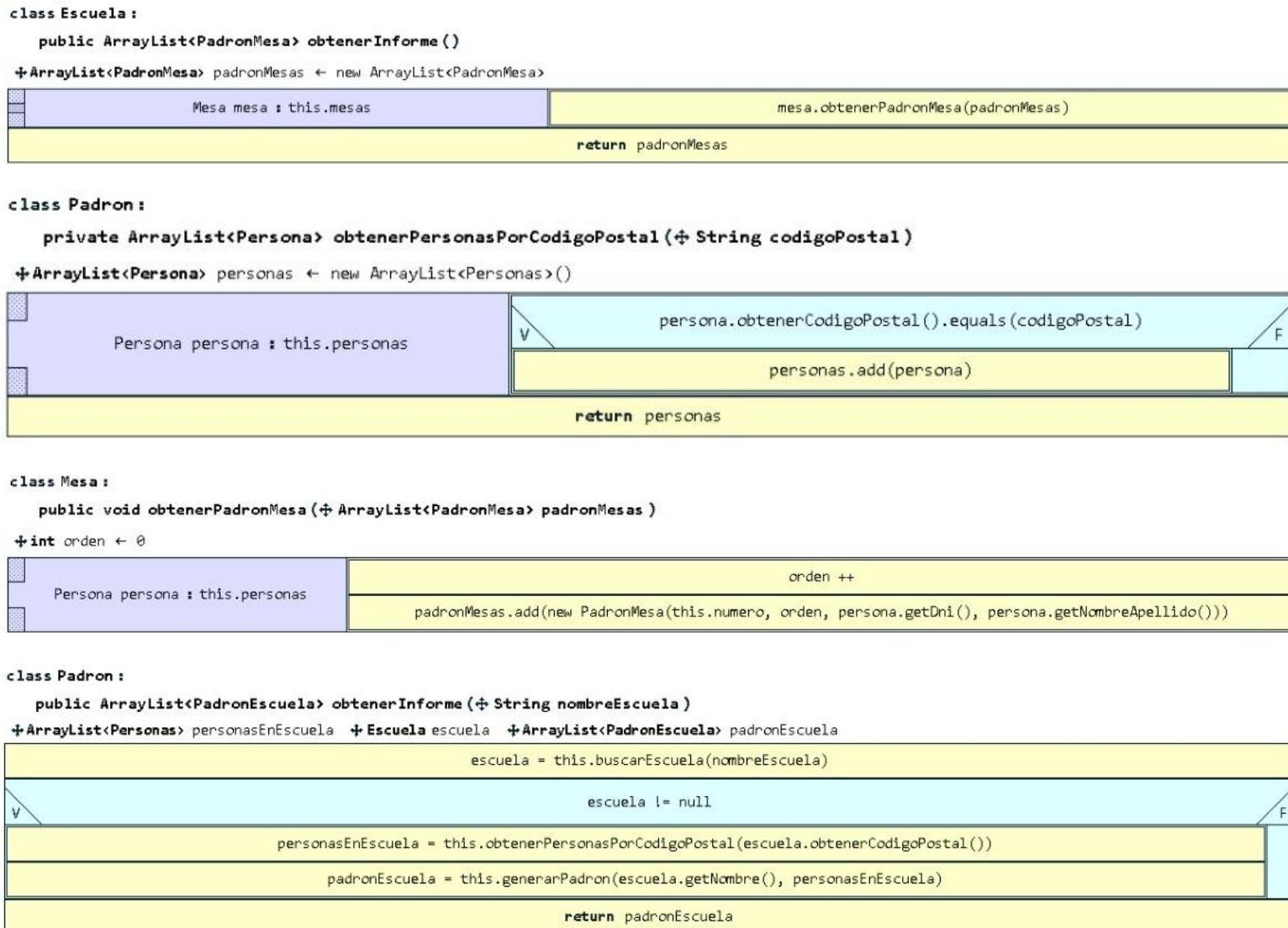
```
class Empresa :  
    public void reincorporarPaquetes()  
    {  
        int pos = 0; RegistroPendiente registro
```



---

```
class Padron :  
    private ArrayList<PadronEscuela> generarPadron (String nombreEscuela, ArrayList<Persona> personasEnEscuela)  
    {  
        ArrayList<PadronEscuela> padronEscuela = new ArrayList<PadronEscuela>()
```





```
class Mesa :  
    public boolean quitarPersona(♦ Persona persona )  
    {  
        int i ← 0   ♦ Persona personaQuitada ← null   ♦ Persona personaActual  
  
        i < this.personas.size() && personaQuitada == null  
        personaActual = this.personas.get(i)  
        V  
        personaActual == persona  
        personaQuitada = this.personas.remove(i)           F  
        this.quitarPresidente(persona)                   i ++  
        return (personaQuitada != null)  
    }  
}
```

