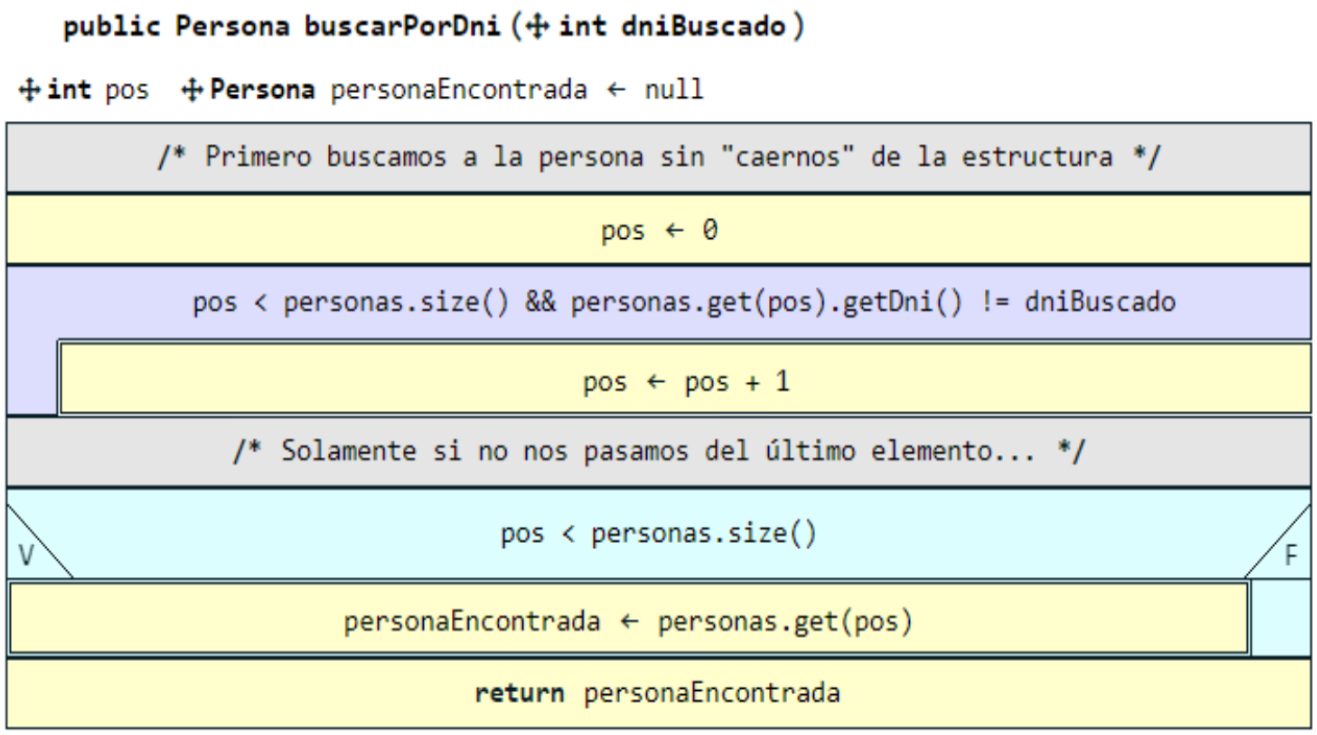
Entender que hace es entender quien lo hace

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 preguntas 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

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

y que ellos los devuelvan y los borren de cada array



Buscar

```
class MusicApp :
    private Usuario buscarUsuario (✣ String email )
✣usuario existeUsuario ← null   ✣int pos ← 0
pos < usuarios.size() && existeUsuario == null
    usuarios.get(pos).getEmail() == email
V                                                F
    existeUsuario ← usuarios.get(pos)   |   pos ← pos + 1
return existeUsuario
```

Buscar más performante

```
class Empresa :
    public Carrera buscarCarrera (✣ String fecha )
✣int pos ← 0   ✣Carrera existeCarrera ← null
pos < carreras.size() && carreras.get(pos).getFecha() != fecha
    pos ← pos + 1
carreras.size() > pos
V                                                F
    existeCarrera ← carreras.get(pos)
return existeCarrra
```

Buscar usando como resultado un metodo

```
class Empresa :
    public Boolean altaEmpleado (✣ String nombre , ✣ String dni , ✣ int nroOficina )
✣int i ← 0   ✣int legajo ← 0   ✣boolean rta ← false
!buscarEmpleadoGral(dni)
V                                                F
    legajo = calcularLegajo()
    i < areas.size() && !areas.get(i).buscarOficina(nroOficina)
        i++
    i < areas.size()
V                                                F
    areas.get(i).altaEmpleado(nombre, dni, legajo)
    rta ← true
return rta
```

Dar de alta en una lista y buscar que no exista

```
class MusicApp :
    public boolean altaCancion (✣ String nombreCancion , ✣ String autor , ✣ String genero )
✣ Cancion existeCancion ← null   ✣ boolean alta
existeCancion ← buscarCancion(nombreCancion, autor)
buscarCancion(nombreCancion, autor) == null
V
F
cancion ← new Cancion(nombreCancion, autor, genero)
canciones.add(cancion)
alta ← true
alta ← false
return alta
```

```
class ORTFlix :
    public RtaAlta darDeAlta (✣ int dni , ✣ Servicio tipoServicio )
✣ RtaAlta respuesta
buscarCliente(dni, deudores) == null
V
F
buscarCliente(dni, clientes) == null
V
F
cliente ← new Cliente(dni, tipoServicio)
clientes.add(cliente)
respuesta ← RtaAlta.ALTA_OK
respuesta ← RtaAlta.CLIENTE_EXISTENTE
respuesta ← RtaAlta.CLIENTE_DEUDOR
return respuesta
```

Listar o recorrer una colección

```
class Usuario :
    public void listarCanciones ()
S
"Listado de canciones escuchadas"
Cancion cancion : historial
S
mostrarNombreCancion()
```

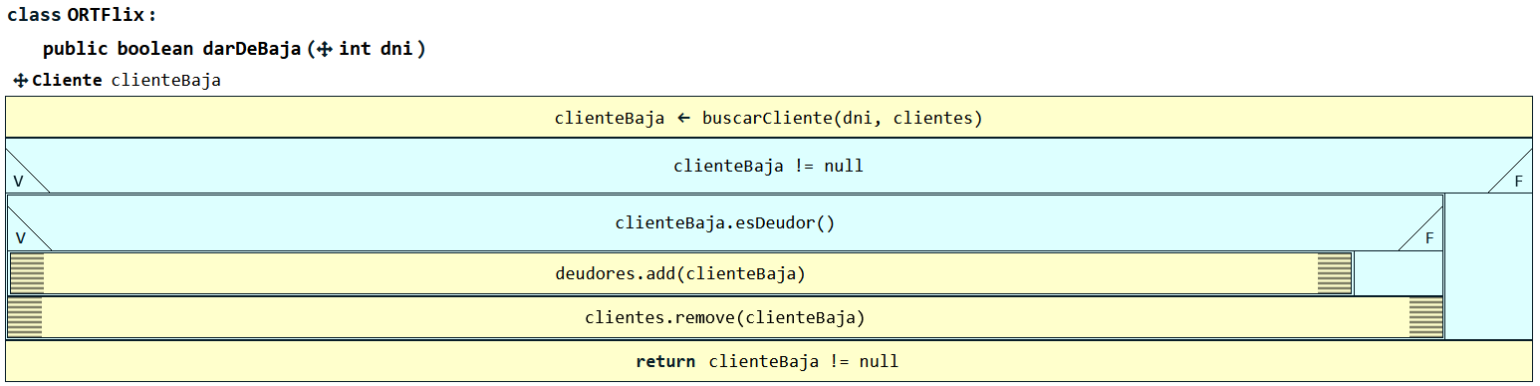
Constructor con variable inicializada y un array

```
class Cancion :
    public  Cancion (✣ String nombreCancion , ✣ String apellido , ✣ int edad , ✣ Categoria categoria )
✣ int contCanciones
this.email ← email
this.apellido ← apellido
this.edad ← edad
this.categoria ← categoria
this.contCanciones ← 0
this.historial ← new ArrayList<Cancion>()
```

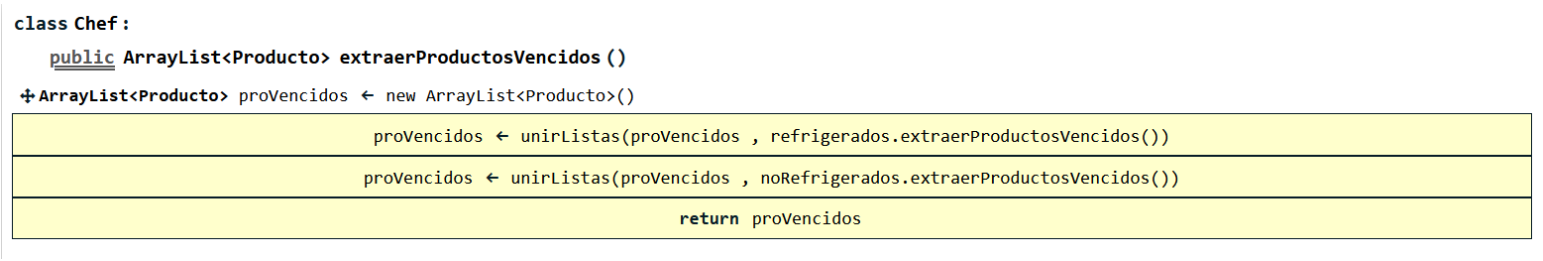
Contructor

```
class PilotosPorCarrrera :
    public  PilotosPorCarrera (✣ String fecha , ✣ int cant )
this.fecha ← fecha
this.cant ← cant
```

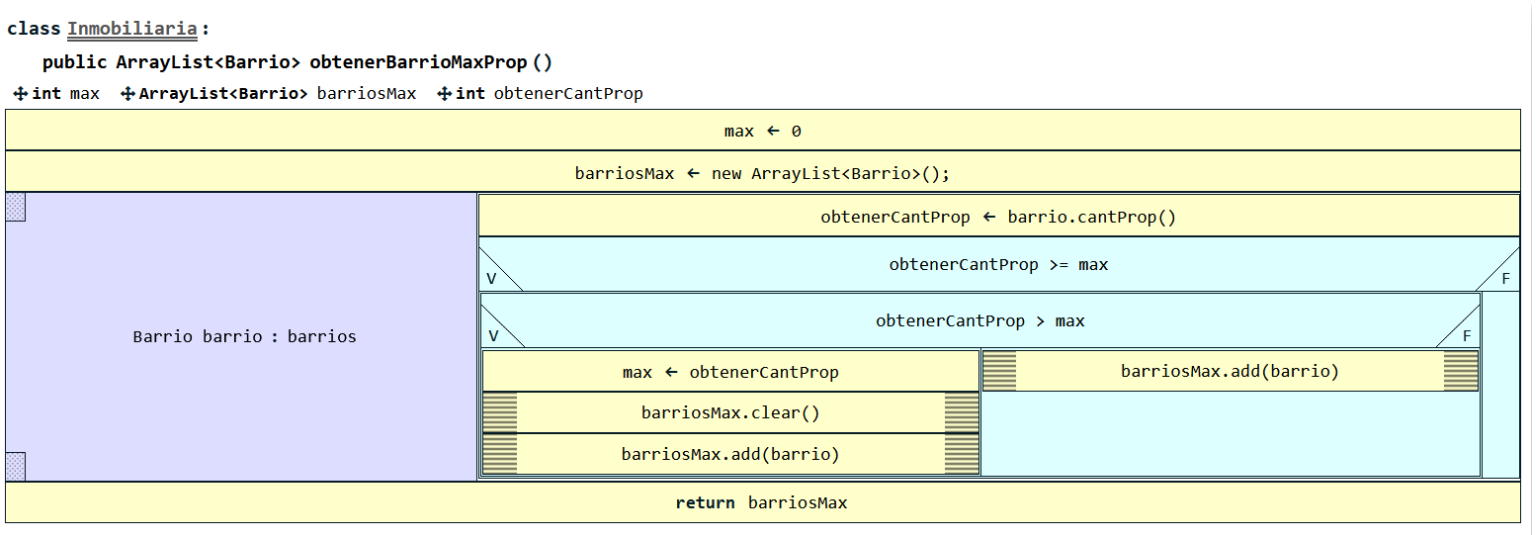
Dar de baja



New ArrayList para devolver



Buscar un máximo agregando a una lista



toString (un poco largo)

```
class Automovil :
    public String toString ()
✣String salidaString
```

salidaString = "marca: " + this.marca + " modelo " + this.modelo + " Capa tanque: " + this.capacidadTanque + "Cant. comb: " + this.cantidadCombustible
salidaString = salidaString + " Rendimiento km por litro: " + this.rendimientoPorLitro
salidaString = salidaString + " Capacidad tanque disponible: " + this.espacioDisponible() + " Poco comb: " + this.pocoCombustible()
return salidaString

Crear un array y agregar ogetos del tipo del array

```
class Empresa :
    public ArrayList<PilotosPorCarrera> pilotosPorCarrera ()
✣ArrayList<PilotosPorCarrera> listaPilotos ← new ArrayList<PilotosPorCarrera>()
```

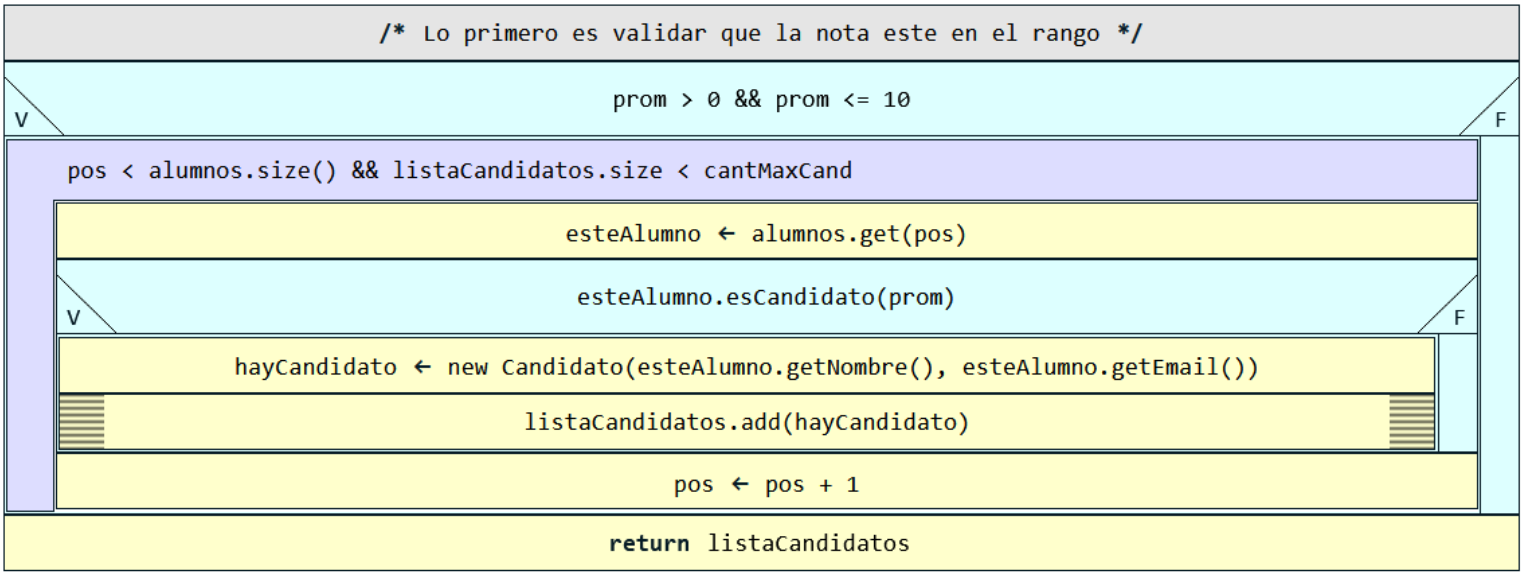
Carrera carrera : carreras	listaPilotos.add(new PilotoPorCarrera(carrera.getfecha(), carrera.cantPilotos()))
return listaPilotos	

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

Piloto pilotoleido : pilotos	piloto.cantidadVueltas() >= CANT_VUELTAS_MIN	V / F
	piloto.calcularPromedio() <= segundos	V / F
	listaPilotos.add(new Piloto(piloto))	
return listaPilotos		

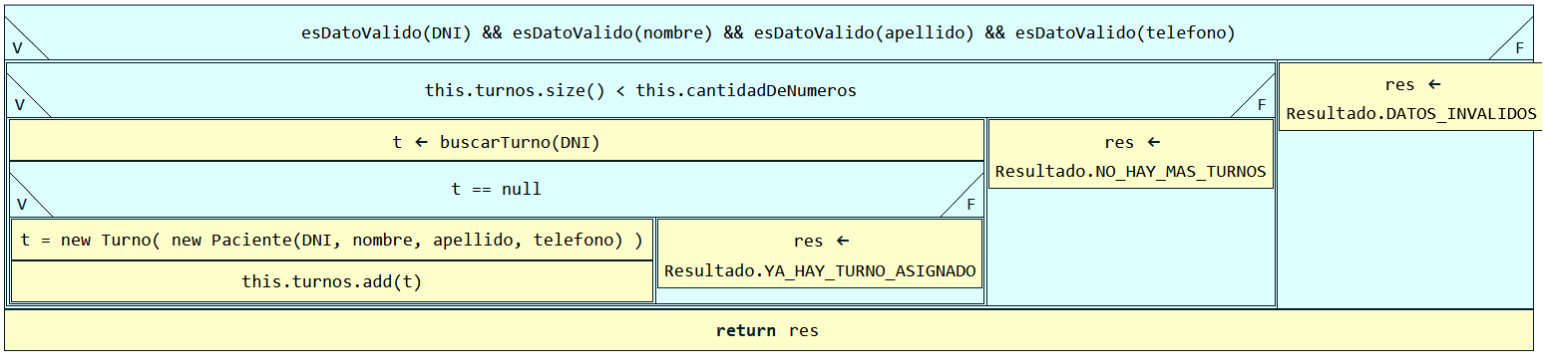
Crear una lista con un tope de máximo de objetos (ojo el size cuando esta en 19, agrega el elemento 20 por eso debe ser posición < size())

```
class Carrera :
    public ArrayList<Candidato> obtenerCandidatos( ✣ double prom )
✣ final int cantMaxCand ← 20   ✣ ArrayList<Candidato> listaCandidatos ← new ArrayList<>()   ✣ int pos ← 0
✣ Candidato hayCandidato   ✣ Alumno esteAlumno
```



Validar datos de ingreso

```
class Agenda :
    public Resultado registrarTurno (✣ String DNI , ✣ String nombre , ✣ String apellido , ✣ String telefono )
✣ Resultado res ← Resultado.TURNO_CONFIRMADO   ✣ Turno t
```




```
class Agenda :
    private static boolean esDatoValido (✣ String laCadena )
                                        return laCadena != null && !laCadena.isEmpty()
```

```
class Agenda :
    private static boolean esDatoValido (✣ int elNumero )
                                        return elNumero > 0
```

Valida que una cadena de String no sea null o este vacia

```
class Validador :
    public static boolean esCadenaValida (+ String laCadena )
        return laCadena != null && !laCadena.isEmpty()
```

devulve una listaa

```
public ArrayList<Elemento> obtenerDisponibilidadJuegos ()
+ArrayList<Elemento> lista ← new ArrayList<>()
Juego j :
this.juegos
    String nom ← j.getNombre()
    int lugares ←
    j.getLugaresDisponibles()
    lista.add( new Elemento(nom, lugares) )
return lista
```

```
public int getLugaresDisponibles ()
+int lugares ← 0
Mesa m : this.mesas
    lugares += m.getLugaresDisponibles()
return lugares
```

```
public int getLugaresDisponibles ()
    return cantMaxParticipantes - participantes.size()
```

```
public boolean repartirCartas ()
+boolean sePudo
sePudo ← jugadoresSuficientes() && alcanzanCartas()
sePudo
V
F
repartir()
return sePudo
```

```
private boolean jugadoresSuficientes ()
return this.participantes.size() >= CANT_JUG_MIN
```

```
private boolean alcanzanCartas ()
return
mazoCartas.size() >= CARTAS_A_REPARTIR * participantes.size()
```

```
private void repartir ()
int i ← 0 , CARTAS_A_REPARTIR - 1 , 1
Participante p : this.participantes
p.agregarCarta( mazoCartas.remove(0) )
```

```
public Resultado acomodarJugador (+ String nomJugador , + int edad , + String nomJuego )
+Resultado res ← Resultado.ASIGNACION_OK   +Juego juego
/* ¡VALIDAR LOS PARÁMETROS! */
juego ← buscarJuego(nomJuego)
juego != null
V
F
juego.getLugaresDisponibles() > 0
V
F
res ←
Resultado.JUEGO_NO_ENCONTRADO
Participante p ←
new Participante(nomJugador, edad)
res ←
Resultado.SIN_DISPONIBILIDAD
juego.agregarParticipante(p)
return res
```

```
private Juego buscarJuego (+ String nomJuego )
+int i ← 0   +Juego j ← null
i < this.juegos.size() && this.juegos.get(i).getNombre() != nomJuego
    i++
i < this.juegos.size()
V                                                                   F
    j ← this.juegos.get(i)
return j
```

```
public void agregarParticipante (+ Participante p )
+int lugaresMin ← 12312312312   +Mesa laMesa ← null
Mesa m : this.mesas
    int lugares ← m.getLugaresDisponibles()
    lugares > 0 && lugares < lugaresMin
    V                                                               F
        lugaresMin ← lugares
        laMesa ← m
return lugares
```

buscar

```
class LaClase :
    public Persona buscarPersona (+ Persona personaABuscar )
+Persona persona = null   +int i = 0
i < personas.size() && persona == null
    personas.get(i) == personaABuscar
    V                                                               F
        persona ← personas.get(i)                                   i++
return persona
```

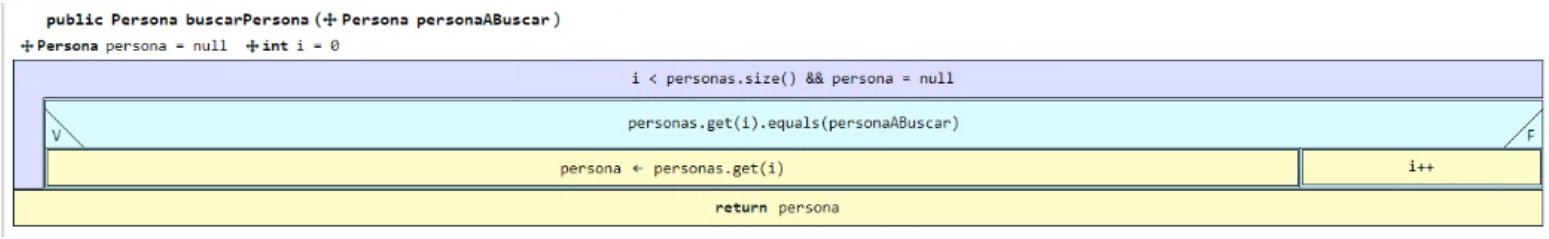
buscar por string

```
public Persona buscarPersona (+ String personaABuscar )
+Persona persona = null   +int i = 0
i < personas.size() && persona == null
    personas.get(i).getNombre().equals(personaABuscar)
    V                                                               F
        persona ← personas.get(i)                                   i++
return persona
```

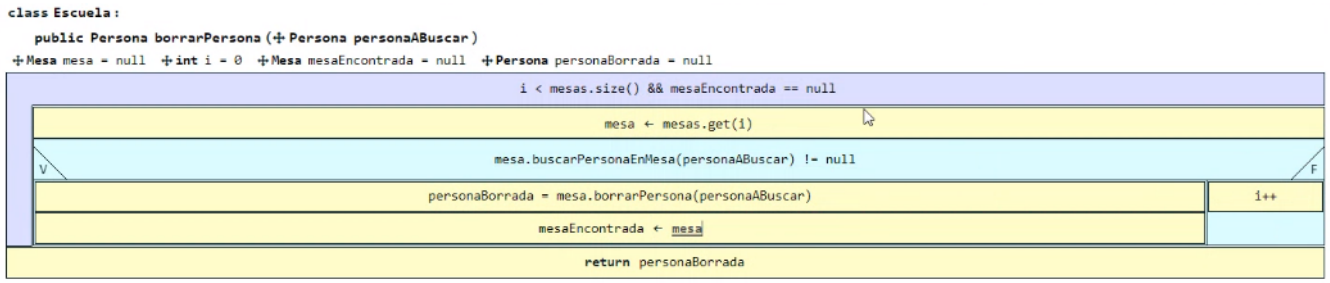
buscar por objeto y con negacion

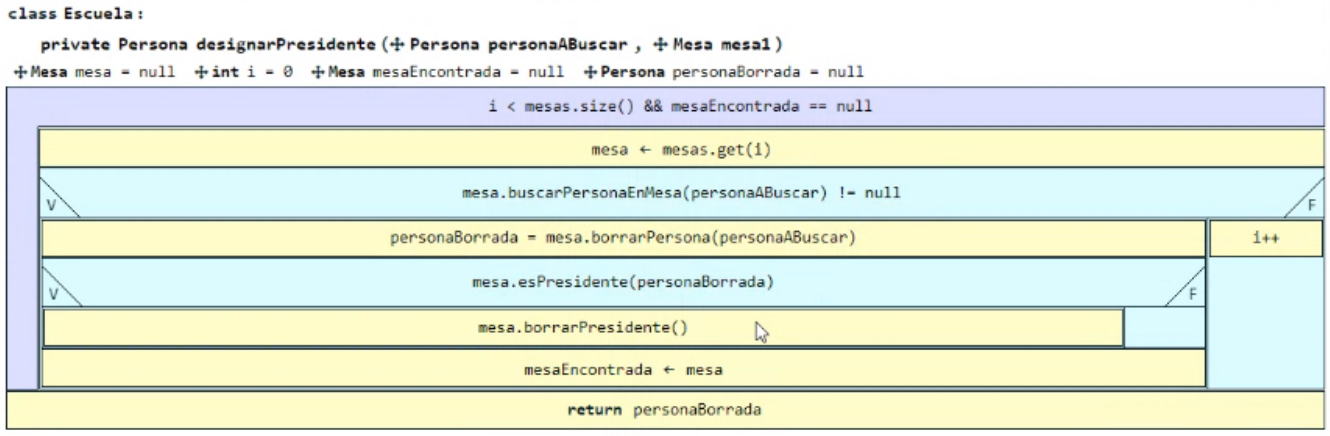
```
public Persona buscarPersona (+ Persona personaABuscar )
+Persona persona = null   +int i = 0
i < personas.size() && !personas.get(i).equals(personaABuscar)
    i++
    i < personas.size()
    V                                                               F
        persona ← personas.get(i)
return persona
```

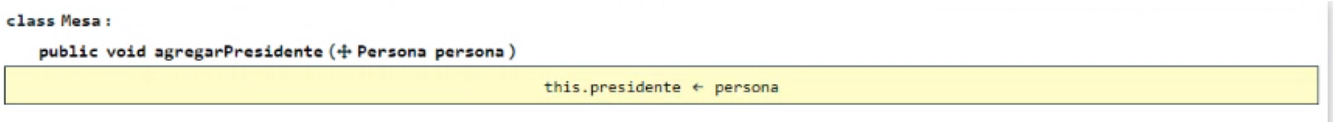
tambien este

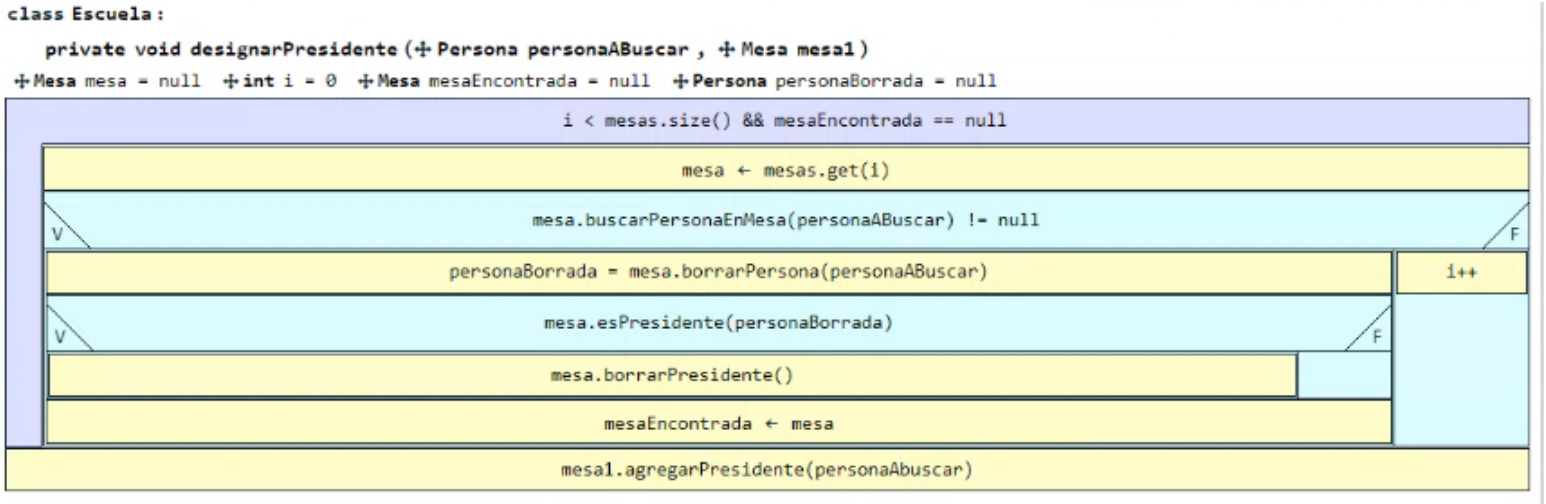


borrar un objeto que esta dentro de un array dentro de otro array









```
class Escuela :
    private void designarPresidente (✣ Persona personaABuscar , ✣ Mesa mesa1 )
✣Mesa mesa = null  ✣int i = 0  ✣Mesa mesaEncontrada = null  ✣Persona personaBorrada = null
primero busco si existe la mesa
i < mesas.size() && mesaEncontrada == null
    /* recorro las mesas y si encuentro la persona la borro */
    mesa ← mesas.get(i)
    mesa.buscarPersonaEnMesa(personaABuscar) != null
        personaBorrada = mesa.borrarPersona(personaABuscar)
        /* si la borre pregunto si tambien es presidente para borrarla */
        mesa.esPresidente(personaBorrada)
            mesa.borrarPresidente()
        mesaEncontrada ← mesa
    /* ahora agrego en la mesa que me piden al presidente y tambien lo agrego como votante */
    i++
mesa1.agregarPresidente(personaAbuscar)
mesa1.agregarVotante(personaABuscar)
/* el metodo agregar votante tambien debe validar si ya existe entre sus votantes, si es asi no lo agrega */
```

Recorrido de una lista con while

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

```
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() && personaEncontrada == null
    personas.get(pos).getDni() != dniBuscado
    V                                        F
    pos ← pos + 1 | personaEncontrada ← personas.get(pos)
return personaEncontrada
```

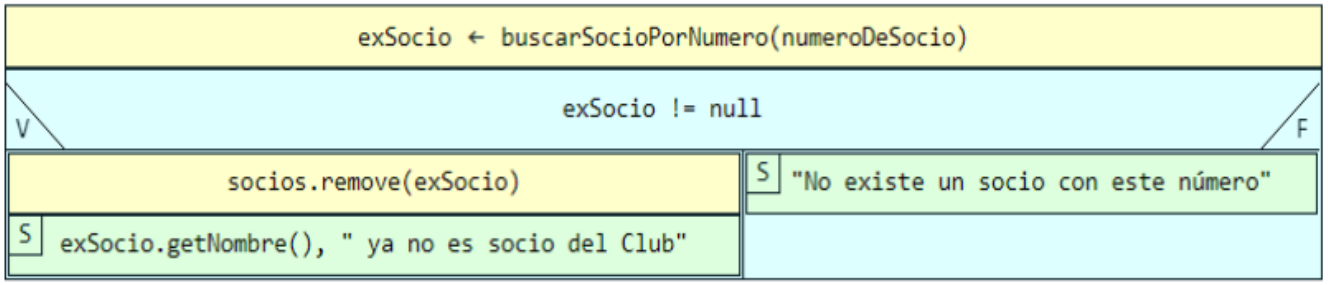
```
public void vaciarColeccion ()
!coleccion.isEmpty()
    verQueHago(coleccion.remove(0))
```

```
class Club :
    public void asociar (✣ String nombre )
✣Socio socio ← null
socio ← buscarSocioPorNombre(nombre)
socio == null
V                                   F
socio ← new Socio(nombre)           S "Este socio ya existe"
socios.add(socio)
S "Se ha dado de alta al socio numero ", socio.getNumero()
```

class Club :

public void darDeBaja (✣ int numeroDeSocio)

✣ **Socio** exSocio ← null



class Club :

private ArrayList<Integer> obtenerVitalicios ()

✣ **ArrayList<Integer>** nrosVitalicios