

Fundamentos de Programación

No se puede usar una variable que no existe

Función MOD(%): Devuelve el resto de la división de dos números. Por ejemplo 7 MOD 3 devolverá 1, ya que al realizar la división entera de 7 entre 3 se obtiene 2 como cociente y 1 como resto. (7 % 3)

```
estaEnAmbosGrupos ← esDeUnSoloDigito && esImpar  
* noEstaEnNinguno = !esDeUnSoloDigito && !esImpar */  
de Morgan negamos la expresion anterior para saber si n  
noEstaEnNingunGrupo ← !(esDeUnSoloDigito || esImpar)
```

negar toda la expresión le ponemos un ! antes y ponemos todo dentro de () y cambiamos el símbolo de adentro sea || a && y && a ||

```
en Java: int mayor=Integer.MIN_VALUE;
```

Parcial:

- todo dato que se ingresa se debe validar
- El promedio no siempre se saca dentro del ciclo, SE HACEN LAS SUMAS DENTRO DEL CICLO Y EL PROMEDIO SE SACA AFUERA DEL CICLO
- Si se da a entender que el ciclo se ejecuta AL MENOS UNA VEZ es un **DO WHILE**
- Si puede que nunca entre al ciclo es un WHILE

Segundo parcial

entender que hace es entender quien lo hace

- depende de la clase que tiene el atributo debe tener el método que pueda modificar o mostrar el mismo atributo
- los getter y setter no son necesarios a menos que se deban utilizar o lleven lógica
- las relaciones de uso se ponen cuando un objeto conoce a otro pero no lo tiene dentro de sus atributos,
- si necesito algo de una lista debo hacer un método que lo informe
- cada método debe resolver una única cosa y si tiene que resolver otras cosas no debe tener toda la lógica dentro si no que tiene que estar modularizado en otros métodos
- cuando recibimos datos de otro método o externos se debe validar siempre
- toda clase que tiene una lista se debe hacer el constructor inicializando la lista

-STATIC : es que le pertenece a la clase y no a un objeto → private static int precio; → Clase.precio

-Una clase A que tiene una lista de tipo clase B la cual tiene una lista de tipo clase C. la clase C es responsable de mostrar sus atributos, la clase B es capaz de llamar a la clase C y decirle que muestre los atributos que ella quiera, y la Clase A puede recorrer cada uno de los objetos de la clase B y pedirle que muestre lo que ella quiera... una clase Fábrica, Sucursal, Instrumento. Fabrica tiene una lista de Sucursal, Sucursal tiene una lista de instrumentos; instrumentos se puede mostrar a el mismo, Sucursal puede recorrer y ver todos los instrumentos que tiene en su lista y Fábrica puede recorrer todas sus sucursales y pedirle que muestre sus instrumentos

-MODELAR SIEMPRE LO QUE NECESITE

Interpretación de enunciado: identificar correctamente clases, atributos, relaciones, enumerados, cardinalidad, relaciones de uso.

Modularización

Reutilización

Encapsulamiento

Asignación de responsabilidad

Abstracción

Atomicidad

Validación de datos (recibidos x parámetro ó ingresados por teclado)

Constructores: No pueden faltar si la clase tiene ArrayList, sino no es necesario salvo que sea pedido expresamente

Getters y Setters: si no tienen ninguna lógica particular, no son necesarios

Final Febrero 2022

Enunciado

Google nos encargó desarrollar un sistema para la instalación y actualización de las aplicaciones de los teléfonos Android.

Este sistema puede recibir por parámetro ninguna, una o más aplicaciones para actualizar y/o instalar según corresponda.

De cada aplicación recibida se conoce su nombre (String), su número de versión (entero), el espacio que ocupa una vez instalada (double) y la versión mínima requerida del sistema operativo del teléfono (entero).

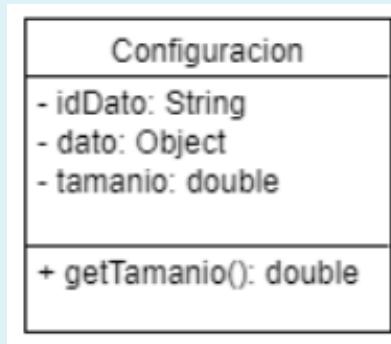
Además, cada aplicación tiene una colección de configuraciones predeterminadas las cuales pueden ir cambiando en la medida que el usuario la utilice. Se entiende por configuración por ejemplo una foto de perfil (avatar), nombre de usuario dentro de la aplicación, etc.

Si la aplicación no existe en el teléfono, se instalará.

Si la aplicación ya existe en el teléfono se debe realizar una actualización. Para esto se debe eliminar la versión existente y luego instalar la nueva. Tomar en cuenta que se deben mantener las configuraciones asociadas a la aplicación.

Antes de instalar o actualizar una aplicación recibida, se debe chequear que el teléfono disponga de espacio suficiente, que la versión del sistema operativo sea igual o superior a la requerida y, en caso de ser una actualización, además se debe chequear que la versión de la aplicación recibida sea mayor a la ya instalada. Si todos los requisitos se cumplen, se podrá llevar a cabo la operación.

De la clase Configuración conocemos el identificador del dato que guarda (String), el dato, más el tamaño que ocupa dicho dato. A continuación presentamos el UML de la clase Configuración.



Nota: El dato del tipo Object no es relevante para la resolución del ejercicio. Indica que el atributo puede recibir cualquier tipo de dato.

Del teléfono se conoce el número de versión del sistema operativo y la capacidad de espacio total de fábrica en el que pueden instalar las aplicaciones.

Se pide:

- Confeccionar el diagrama UML que describe el escenario del enunciado incluyendo los atributos de cada clase y los métodos a desarrollar y aquellos que creas conveniente.
- Desarrollar el método calcularEspacioOcupado() que no recibe parámetros y devuelve un valor double que indica la cantidad de espacio ocupado por las aplicaciones instaladas.
- Desarrollar el método gestionarAplicaciones(...) que recibe las aplicaciones para instalar o para actualizar y hace su tarea según lo descripto más arriba devolviendo un resultado posible por cada una de las aplicaciones.

- INSTALACION_OK: la aplicación se instaló en forma correcta.
- ACTUALIZACION_OK: la aplicación se actualizó en forma correcta.
- ESPACIO_INSUFICIENTE: el teléfono no dispone de espacio suficiente para instalar o actualizar dicha aplicación.
- VERSION_INCORRECTA: la versión del sistema operativo es menor a la requerida por la aplicación.
- VERSION_NO_SUPERIOR: la versión de la aplicación recibida no es superior a la ya existente en el teléfono.

Los diagramas forman parte de las herramientas de la programación, se usan en un lenguaje *único* que no da lugar a ninguna confusión o mala interpretación.

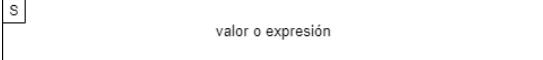
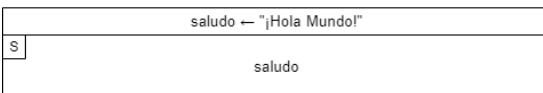
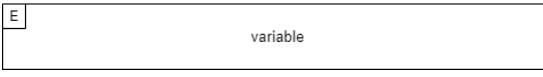
1. Bloques de instrucciones:

- Instrucciones básicas con variables: *asignación* es la instrucción que permite poner un determinado valor como contenido de una variable. Primero necesitamos crear o declarar la variable

variable ← valor o expresión	int cantidad;
cantidad ← 0	cantidad = 0;
saludo ← "¡Hola Mundo!"	string saludo = "Hola mundo"
promedio ← acumulado / cantidad	double promedio; promedio = acumulado / cantidad

- Instrucciones de entrada o salida: Salida o Entrada (La diferencia entre uno y otro es que *las operaciones de entrada solo pueden tener como elemento una variable*, pues es el único elemento que puede contener un *dato*)
- Tienen en su esquina superior derecha o izquierda una letra que identifica el tipo de operación (si lo vemos en nuestro idioma será una "E" para entrada y "S" para las salidas).

Salida: pueden mostrar tanto *valores* (sean *literales*, *constantes* o *variables*) u *operaciones* que devuelvan un valor

	En java System.out.println()
<pre>String saludo; saludo ← "¡Hola Mundo!"</pre> 	<pre>string saludo = "Hola mundo" System.out.println(saludo)</pre>
<pre>[tipo de dato] variable;</pre> 	<p>Entrada: en las operaciones de entrada siempre el contenido es una variable.</p> <p>En java Scanner input = new Scanner(System.in);</p> <p>Es recomendable no utilizar el scanner de esta manera para capturar números. La instrucción equivalente que recomendamos es:</p> <pre>numeroEntero = Integer.parseInt(input.nextLine());</pre> <p>También para los números reales haremos:</p> <pre>double numeroReal; numeroReal = Double.parseDouble(input.nextLine());</pre>

- Instrucciones de control de flujo:

Condicional simple: conocido como **if**, permite elegir entre dos caminos posibles a partir del resultado de una condición lógica (cuyo valor sólo puede ser verdadero o falso).

```
→ switch(variable o expresión enumerable) {
    case A: ...lo que hago si vale A: break;
    case B: ...lo que hago si vale B: break;
    default: ...lo que hago si es cualquier otro valor;
}
```

variable o expresión lógica

V
lo que hago si la condición es verdadera

F
lo que hago si la condición es falsa (podría no hacer nada)

a > b

V
S
"el valor de a es mayor que el de b"

F
S
"el valor de b es mayor o igual que el de a"

Condicional múltiple: conocido como *switch*, permite elegir entre varios caminos a partir de un valor *enumerable*. La estructura del *switch* es...

variable o expresión enumerable

A	B	default
lo que hago si vale A	lo que hago si vale B	lo que hago si es cualquier otro valor
break	break	

Si bien el *break* no forma parte de la estructura del *switch*, es necesario al final de cada *case*, excepto el último, para que no se sigan ejecutando las instrucciones por caída. Esto quiere decir que si no ponemos el *break* en la primera opción y el valor de la expresión es A, se ejecutarán los bloques de A y de B, consecutivamente.

Instrucciones repetitivas o ciclos (*while*, *do-while* y *for*).

Los *ciclos* permiten que una instrucción (o un grupo de dos o más instrucciones) se ejecute una y otra vez mientras se den las condiciones deseadas.

A éstos agregamos, para la POO el ciclo *for-each*, que utilizamos para recorrer *colecciones de datos* o cualquier otro elemento que sea iterable.

While (ciclo 0-n)

Se lo llama así porque puede ser que no haya posibilidad de *entrar* al ciclo (porque ya antes de entrar la condición del ciclo no se cumple) y porque, una vez que se entró al ciclo, lo único que permitirá que se salga es que la condición deje de cumplirse. Se caracteriza porque la condición de permanencia

antecede al bloque de instrucciones que conforman el ciclo:

si en una prueba recorre 20 minutos ya no importa que haga el resto ya no está apto

condición de permanencia

instrucción o bloque de instrucciones

variableDeControl ← valor inicial

estado de variableDeControl

proceso(...)

variableDeControl ← valor de actualización

while (condicion_de_permanencia) {instrucción o bloque de instrucciones;}

Esto se traduce en java como:

```
int numeroDeDia;
System.out.print("Ingresa el numero de dia de la semana (1 a 7)");
numeroDeDia = Integer.parseInt(input.nextLine());
while (numeroDeDia < 1 || numeroDeDia > 7) {
    System.out.print("Me parece que te equivocaste. " +
                    "Ingresa el numero de dia de la semana " +
                    "asegurandote de que sea un numero " +
                    "entre el 1 y el 7 (inclusive):");
    numeroDeDia = Integer.parseInt(input.nextLine());
```

Uno de los usos habituales de los ciclos es el ingreso validado de valores

Do-while (ciclo 1-n)

Se lo llama *ciclo 1-n* porque, a diferencia del *while*, su condición de permanencia se

escribe y chequea luego de ejecutar el bloque de instrucciones que conforman el ciclo. Por lo tanto, siempre se entrará al ciclo aunque, al igual que en el caso de *while*, una vez que entró al ciclo sólo se permitirá salir cuando la condición deje de cumplirse.

variableDeControl ← valor
proceso(...)
estado de variableDeControl

El bucle **do while** comprueba la condición después de ejecutar las instrucciones por lo tanto es un **Exit Control Loop** (Salir del bloque de control). Es importante aquí resaltar que no hay dos puntos de actualización de la

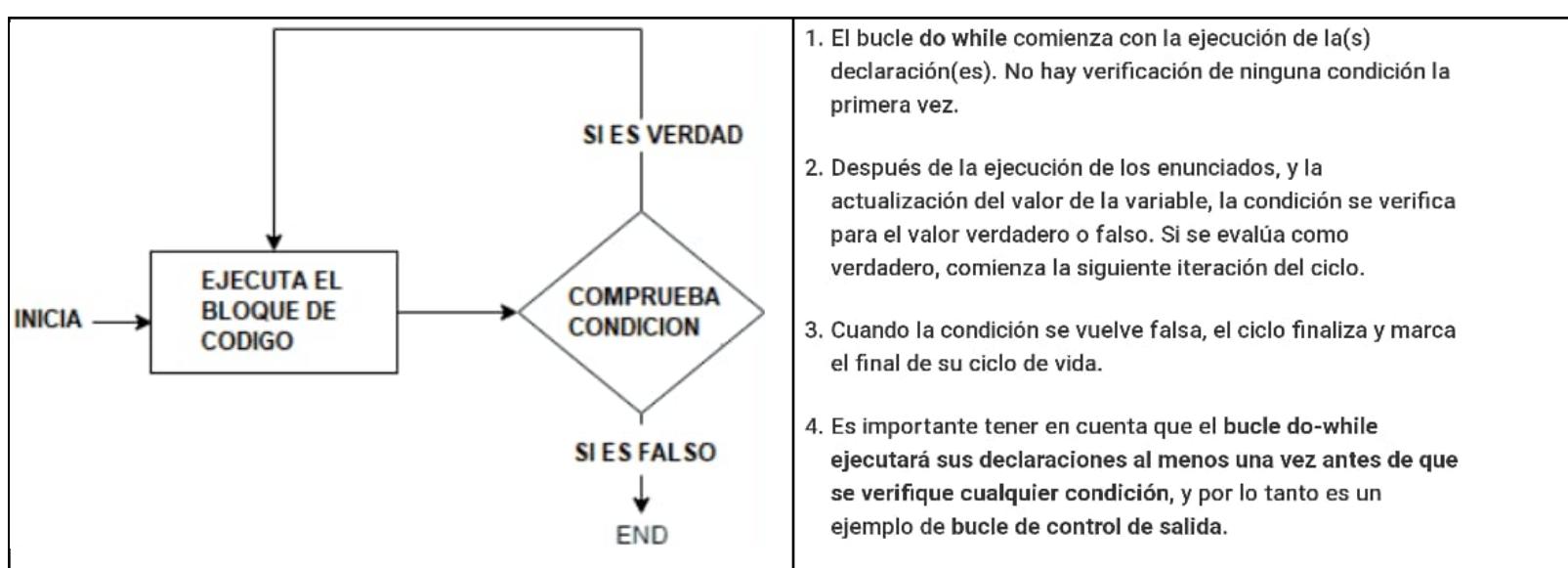
variable de control, y que éste es apenas se ingresa al ciclo, ni antes ni después.

Dado que el dato se carga apenas antes de ser procesado, y dentro del ciclo, en caso de desear un tratamiento especial en caso de error implicaría que usemos un *if* para mostrar el problema. Para hacer eso, aconsejamos usar el ciclo *while*.

For (ciclo fijo)

Se conoce al *for* como *ciclo fijo* porque la cantidad de veces que se ciclará es conocida *antes de entrar al ciclo*, siempre y cuando no se altera artificialmente el valor de la variable de control.

variable = valorInicio, valorFinal, paso	bloque de instrucciones del ciclo
------------------------------------------	-----------------------------------

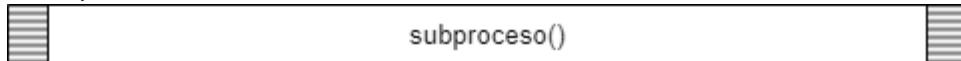


<pre> graph LR A[INICIA] --> B{COMPRUEBA CONDICION} B -- SI ES VERDAD --> C[EJECUTA EL BLOQUE DE CODIGO] C --> B B -- SI ES FALSO --> D[] </pre>	<ul style="list-style-type: none"> El while comienza con la verificación de la condición. Si se evalúa como verdadero, las instrucciones del cuerpo del bucle se ejecutan; de lo contrario, se ejecuta la primera instrucción que le sigue al bucle. Por esta razón, también se llama bucle de control de entrada. Una vez que la condición se evalúa como verdadera, se ejecutan las instrucciones en el cuerpo del bucle. Normalmente, las declaraciones contienen un valor de actualización para la variable que se procesa para la siguiente iteración. Cuando la condición se vuelve falsa, el ciclo finaliza y marca el final de su ciclo de vida. 	<p>Un bucle While es una sentencia de control de flujo que permite que el código se ejecute repetidamente en función de una condición booleana. Este se considera como una instrucción if repetitiva</p>
---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

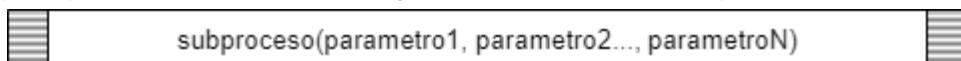
<pre> graph LR A[INICIA] --> B[INICIALIZACIÓN] B --> C{COMPRUEBA CONDICION} C -- SI ES VERDAD --> D[EJECUTA EL BLOQUE DE CODIGO] D --> C C -- SI ES FALSO --> E[END] F[AUMENTA / DISMINUYE] --> C F --> B </pre>	<ol style="list-style-type: none"> Condición de inicialización: Aquí, inicializamos la variable en uso. Marca el inicio de un ciclo for. Se puede usar una variable ya declarada o se puede declarar una variable, solo local para el bucle. Condición de prueba: se usa para probar la condición de salida de un bucle. Debe devolver un valor booleano. También es un bucle de control de entrada cuando se verifica la condición antes de la ejecución de las instrucciones de bucle. Ejecución de instrucción: una vez que la condición se evalúa como verdadera, se ejecutan las instrucciones en el cuerpo del bucle. Incremento/Decremento: se usa para actualizar la variable para la siguiente iteración. Terminación de bucle: cuando la condición se vuelve falsa, el bucle termina marcando el final de su ciclo de vida. 	<p>El bucle for consume la inicialización, la condición y el incremento/decremento en una línea, proporcionando así una estructura de bucle más corta y fácil de depurar.</p>
---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Programación Orientada a Objetos

Llamado a subprocessos: nacen de la necesidad de reutilizar porciones de código. El único componente de una llamada es el nombre del subprocesso seguido de un par de paréntesis.

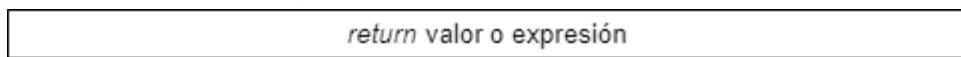


si ese subprocesso necesita recibir parámetros, éstos se incluirán dentro de los paréntesis, uno detrás del otro y separados por comas, y siempre respetando el orden y el tipo de cada uno de ellos según la declaración del subprocesso



Return

Todo método que no sea del tipo `void` deberá devolver un valor a través de la cláusula `return`.



Por ejemplo, un `getter()` devuelva el valor del campo `nombre` sería así:

```

public String getNombre() {
    return this.nombre;
}
    
```

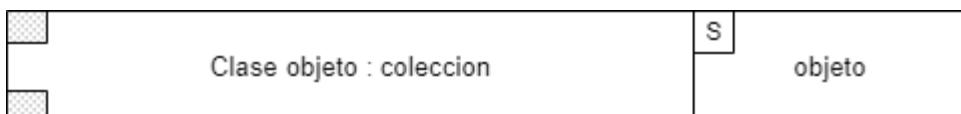
No es necesario que el valor de retorno sea una variable. Por ejemplo, el método `doble(int valor)` devuelve valor recibido multiplicado por 2:

```

public int doble(int valor) {
    return valor * 2;
}
    
```

For-each

El `for-each` (literalmente "*para cada uno [de los elementos de...]*") es el miembro más nuevo de los ciclos, y nació especialmente para recorrer colecciones de datos.



Atributos

Los atributos modificadores de un campo de clase pueden ser:

- static** Indica que se trata de un *atributo o campo de clase*, y no *de instancia*. Cuando es atributo de clase, todas las instancias de la clase comparten el mismo valor.
- final** Una vez asignado su valor no puede ser cambiado. Se comporta como una *constante*. En este caso, su nombre se escribe completamente en mayúsculas, separando cada palabra de su nombre, si éste es completo, con un guión bajo ("_").

Métodos

Los *métodos* de una clase responden a la siguiente declaración y estructura:

```
visibilidad [abstract] [final] tipoRetorno nombreMetodo ([listaArgumentos]) {
    // instrucciones;
```

- visibilidad** Siendo también miembros de la clase como los campos, los métodos tienen las mismas características en cuanto a la *visibilidad*. Puede ser *public*, *protected*, *private* o no declararse, por lo cual será visible desde dentro del *package*.

- abstract** Se declara como *abstract* (*abstracto*) aquel método que solamente se *declara*, y cuyo código deberá ser implementado en alguna *clase derivada*. Con tener un solo método abstracto, propio o heredado, la clase también deberá ser *abstracta*.

- final** Significa que este método no podrá ser sobreescrito por ninguna clase derivada.

<i>tipoRetorno</i>	Tipo de dato a devolver por el método. Puede ser <i>void</i> (y el método no tendrá la cláusula <i>return</i>) y de algún tipo, sea este nativo, el nombre de una clase o cualquier otra definición de tipo de dato más compleja (colecciones, etc.)
<i>nombreMetodo</i>	Nombre del método, y se lo usará para invocarlo acompañado de paréntesis y, si es necesario, con los parámetros correspondientes. Debe escribirse en formato <i>camelCase</i> .
<i>listaArgumentos</i>	Enumeración de cada uno de sus argumentos o parámetros. De cada uno se tiene su tipo de dato y su nombre, y se separan entre sí con una coma.

Luego van todas sus instrucciones encerradas entre llaves, incluyendo la declaración y definición de sus variables locales, si las tuviese. Si el método no es de tipo *void*, su última instrucción debería ser un *return* acompañado por el valor final de retorno.

Constructores

Los constructores son los métodos de inicialización de las clases por excelencia. Son tan especiales que no pueden ser invocados salvo con el operador *new* cuando queremos crear una *instancia* de la clase, o en forma especial desde dentro de los mismos constructores utilizando las cláusulas *this* (que veremos enseguida) y *super* (que aún no utilizaremos).

La estructura de un constructor es:

```
visibilidad nombreClase([listaArgumentos]) {
```

```
    instrucciones;
```

En el caso de los constructores, la visibilidad puede ser *public* o *private* (sí, un constructor puede ser privado, más adelante veremos cómo puede usarse).

Una misma clase puede tener ninguno, uno o más constructores, siempre y cuando éstos no tengan la misma *firma* de declaración.

Es posible que un constructor invoque a otro de la misma clase a través de la cláusula *this*. Veamos un ejemplo:

```
public class Producto {
    public Producto() {
        this(0);
```

```

    }
    public Producto(int codigo) {
        this.codigo=codigo;
    }
}
}

```

En este caso, el primer constructor se aprovecha del segundo para setear el campo *código* en cero.

Método main()

Los métodos *main()* son los puntos de acceso al programa desde el exterior, cuando los programas son invocados.

Java no genera ejecutables, pero todo programa debe tener un punto de entrada. Cualquier clase puede tener este método, pero solamente uno de ellos será el punto de entrada y será ejecutado en primer lugar.

Este método debe ser *public* y *static*. Puede recibir argumentos a través del *array de strings args*.

una clase singular no puede tener una lista de su propia clase, entonces otra clase si tiene una lista de objetos de x Clase la puede recorrer y mostrar

Relación de uso

Esta relación se presenta cuando un objeto usa temporalmente a otro objeto para cumplir con su responsabilidad, usualmente el objeto usado sea desechado luego de ser usado

El objeto usado puede ser un parámetro recibido del objeto que lo usa. Con esto queremos decir que esta relación se refleja en el código que por ende se refleja solo en la clase (y quizás por esto sea más visible cuando estamos desarrollando el código de la clase)

Otra forma de que ocurra esta relación es cuando un objeto instancia a otro temporalmente como variable local en un método, también para cumplir con su responsabilidad. En estos casos, también se suele indicar en los diagramas de clase que el objeto que hace uso “instancia” al otro.

La relación de uso nunca va a expresarse como un atributo de la clase.

Métodos del arrayList

- Insertar un elemento (método **add**) .
- Obtener un elemento de una posición determinada (método **get**) .
- Eliminar un elemento (método **remove**) .
- Averiguar su tamaño (método **size**) .
- Preguntarle si está vacía (método **isEmpty**) .
- Descartar todos los elementos (método **clear**)

Extras

CICLOS x CONTADOR y ACUMULADOR

CONTADOR:

- Es una variable de tipo entero, que incrementa o decrementa su valor de forma constante y requiere ser inicializada generalmente en 0 o 1, aunque en realidad depende del problema que se está resolviendo.
- Siempre es una variable.
- Me sirve cuando sé de antemano cuántas veces entró al ciclo. Lo carga el programador de manera fija o el usuario mediante una variable de Entrada.
- Declaró dicha variable fuera del programa
- La inicializo dentro. (es la mejor costumbre).
- Se inicializan dentro del programa pero siempre fuera del ciclo.
- El valor con el que inicializo es a partir de dónde empieza a contar.
- Para que sume en c/vuelta, debo actualizarlo pero esta vez dentro del ciclo pero al final con la cantidad de pasos que quiero que cuente (es decir se le suma o resta un N.^o fijo/constante de pasos). Es importante actualizar el contador xq sino hace un bucle eterno.
- Si termina ahí el programa, puedo hacer una salida para mostrar el resultado x pantalla x fuera del ciclo xq sino repite la salida c/vez que el ciclo pega una vuelta.

EJEMPLO EN SECUENCIA EN CICLO WHILE:

- Inicializo la variable contador → 0 (fuera del while).
- Abro el While y pongo su condición (en este caso en VERDADERO. Si cumple entra al ciclo sino vuelve a preguntar) → “Mientras” : contador <= 40. Va a contar desde el 0 (inicializado) hasta el 40 de la condición (xq dice <=).
- Pongo una Salida dentro del ciclo que muestra el contador y concateno con un espacio entre ” ” para que los N^o no salgan pegados.
- Actualizar el contador asignando una expresión que va a ser: contador → contador + 2 (pasos).
- P/Ilegar al 40, va a contar de 2 en 2. Cuando llegue al 40 va a hacer una vuelta más y como va a sumar 2 se va a convertir en 42 y se pasa. Entonces corta xq no cumple la condición del ciclo que pide hasta 40 y queda clavado ahí.
- Puedo sumar una Salida que debe ir fuera del ciclo como conclusión final; xq sino repite la salida c/vez que el ciclo pega una vuelta.
- Ejecutó

```

public int sumatoriaKmsRecorridos() {
    int acu = 0;
    for (Auto auto : autos) {
        acu = acu + auto.getKmsRecorridos();
        // acu += auto.getKmsRecorridos();
    }
    return acu;
}

public Auto buscarAuto(String patente) {
    Auto autoEncontrado = null;
    int i = 0;
    while (i < cantAutos() && !this.autos.get(i).getPatente().equalsIgnoreCase(patente))
        i++;
    }
    if (i < cantAutos()) {
        autoEncontrado = this.autos.get(i);
    }
    return autoEncontrado;
}

```

B) La explotación del método instrumentosPorTipo que devuelve una lista de instrumentos cuyo tipo coincide con el recibido por parámetro.

```

public ArrayList<Instrumento> instrumentosPorTipo(TipoInstrumento tipo) {
    ArrayList<Instrumento> instEncontrados = new ArrayList<>();
    for (Instrumento instrumento : instrumentos) {
        if (instrumento.getTipo() == tipo) {
            instEncontrados.add(instrumento);
        }
    }
    return instEncontrados;
}

```

La explotación del método porInstrumentosPorTipo que reciba el nombre de una sucursal y retorne los porcentajes de instrumentos por tipo que hay para tal sucursal.

```

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

```

V

F

Linietzky Norberto



¿Se cuantas veces estoy en 1 ciclo?

V

```

/* Ciclo de 1 - n */
/* CICLO FOR */

```

F

/* Ciclo de 0 - n */

/* CICLO WHILE */

/* Ciclo de 1 - n */

/* CICLO DO WHILE */

```

    /* ***** INICIO *****/
    /* ***** DESARROLLO *****/
    /* Ingreso */
    condicion
    /* Proceso */
    /* Ingreso */
    /* ***** FIN *****/

```

```

    /* INICIO GENERAL */
    /* DESARROLLO GENERAL */
    /* INICIO PARTICULAR DE CADA UNO */
    /* DESARROLLO PARTICULAR DE CADA UNO */
    /* Ingreso la condición de fin */
    pregunto por la condición de fin
    /* Ingreso el resto de los campos */
    /* Proceso */
    /* Ingreso la condición de fin */
    /* FIN PARTICULAR DE CADA UNO */

    /* FIN GENERAL */

```

e ← 1 , 5 , paso

```

    /* INICIO GENERAL */
    /* DESARROLLO GENERAL */
    /* Ingreso Letra */
    pregunto x la letra
    /* INICIO PARTICULAR */
    /* DESARROLLO PARTICULAR */
    var ← inicio , fin , paso
    /* Ingreso */
    /* Proceso */
    /* FIN PARTICULAR */
    /* Ingreso Letra */
    /* FIN GENERAL */

```

```

    /* INICIO GENERAL */
    /* DESARROLLO GENERAL */
    /* INICIO PARTICULAR */
    /* DESARROLLO PARTICULAR */
    var ← inicio , fin , paso
    /* INGRESO */
    /* PROCESO */
    /* FIN PARTICULAR */
    /* Pregunta */
    mientras quiera seguir
    /* FIN GENERAL */

```

```

    /* INICIO GENERAL */
    /* DESARROLLO GENERAL */
    /* Ingreso letra */
    condicion
    /* INICIO PARTICULAR */
    /* DESARROLLO PARTICULAR */
    /* Ingreso Libro */
    condicion
    /* Ingreso resto de campos */
    /* Proceso */
    /* Ingreso_Libro */
    /* FIN PARTICULAR */
    /* Ingreso letra */
    /* FIN GENERAL */

```

```

    /* ***** INICIO DE TODOS LOS CURSOS***** */
    /* Mostrar Portada */

    /* Poner variables que acumulan en cero */

    /* ***** DESARROLLO DE TODOS LOS CURSOS***** */

    /* ***** INICIO DE 1 CURSO ***** */
    /* Mostrar Portada */

    /* Poner variables que acumulan en cero */

    /* ***** DESARROLLO DE 1 CURSO ***** */

    /* Ingreso aporte */

    aporte <> -1

    /* Proceso */

    /* Ingreso aporte */

    /* ***** FIN DE 1 CURSO ***** */

    /* ***** FIN DE TODOS LOS CURSOS ***** */

```

P+1, 9, 1

Parcial atletismo

```

+final int T_RECORD = 15 +final double TOPE_PROM = 18 +final int T_TOPE = 20 +final int T_MENOR = 0 +final int T_MAYOR = 100 +final int DIAS = 10
+int dia=1 +int tiempo +boolean apto +int tiempoTotal +double promedio +int tMenor = MAX_VALUE +int diaMenor +boolean aptoRecord = false

```

```

S "Ingrese en tiempo del dia " + dia
E tiempo
    tiempo <= T_MENOR || tiempo >= T_MAYOR
    apto = (tiempo<=T_TOPE)
    apto
    tiempoTotal += tiempo
    V tiempo < tMenor
        tMenor = tiempo
        diaMenor = dia
        dia++
    dia <= DIAS && apto
    V apto
        aptoRecord = (tMenor < T_RECORD)
        promedio = tiempoTotal / DIAS
        V aptoRecord && promedio <= TOPE_PROM
            S "Atleta NO Apto"
            S "Atleta NO APTO"
            S "Esta Apto, el promedio fue " + promedio
            S "El dia con menor tiempo fue " + diaMenor

```

ZOO

```
+final int TANDAS ← 3 +char seguirComiendo +int kgComida +int kgMayorTanda = MIN_VALUE +int tandaMayor +int kgTanda +int kgTotales=0 +double promedio
int i ← 1 , i <= TANDAS , i++
kgTanda = 0
S "Ingrese kg de comida"
E kgComida
kgComida <= 0
kgTanda += kgComida
S "Desea seguir comiendo?"
E seguirComiendo
seguirComiendo!= 'S'
kgTotales += kgTanda
V kgTanda > kgMayorTanda
F kgMayorTanda = kgTanda
tandaMayor = i
promedio = kgTotales / TANDAS
S "El león comio más en la tanda " + tandaMayor + " y comio en esa tanda " + kgMayorTanda + " kilos"
S "Total de alimento recibido " + kgTotales + " promedio por tanda " + promedio
```

Negación

A	$\neg A$
V	F
F	V

Conjunción

A	B	$A \wedge B$
V	V	V
V	F	F
F	V	F
F	F	F

Disyunción

A	B	$A \vee B$
V	V	V
V	F	V
F	V	V
F	F	F

Negación: (NO) != (<>)

Conjunción: && (Y)

Disyunción: || (O)

Condicionalidad: (Si... entonces)

Bicondicionalidad: (Si y sólo si)

Condisionalidad

A	B	$A \rightarrow B$
V	V	V
V	F	F
F	V	V
F	F	V

Bicondicionalidad

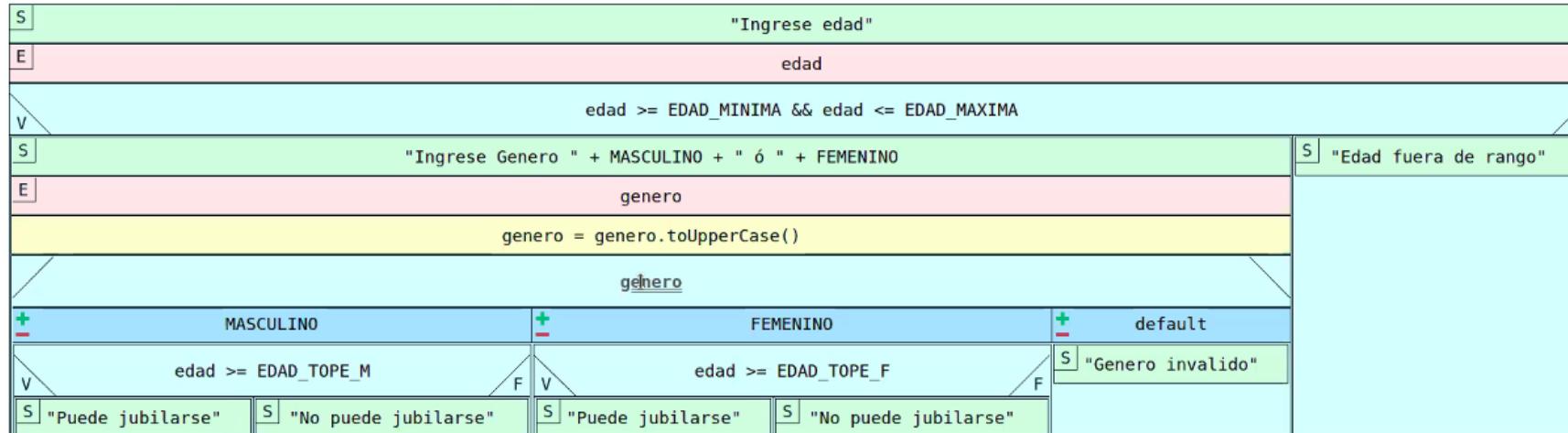
A	B	$A \leftrightarrow B$
V	V	V
V	F	F
F	V	F
F	F	V

```
/* Portada */
S "Programa que muestra el promedio de 5 edades"
E /* Poner variables en cero */
V suma ← 0
F contMayImp ← 0
C /* ***** DESARROLLO **** */
I /* Ingreso */
S "Ingrese una edad"
E edad
F /* Proceso */
V suma ← suma + edad
F (edad > EDAD_MIN) && ((edad % 2) = 1)
C contMayImp ← contMayImp + 1
I /* ***** FIN(suma, CANT, contMayImp) **** */
F /* Calculo el promedio */
C prom ← suma / CANT
I /* muestro el promedio */
S "El promedio es ", prom
S "Y la cantidad de edades impares y mayores de ", EDAD_MIN, " es ", contMayImp
```

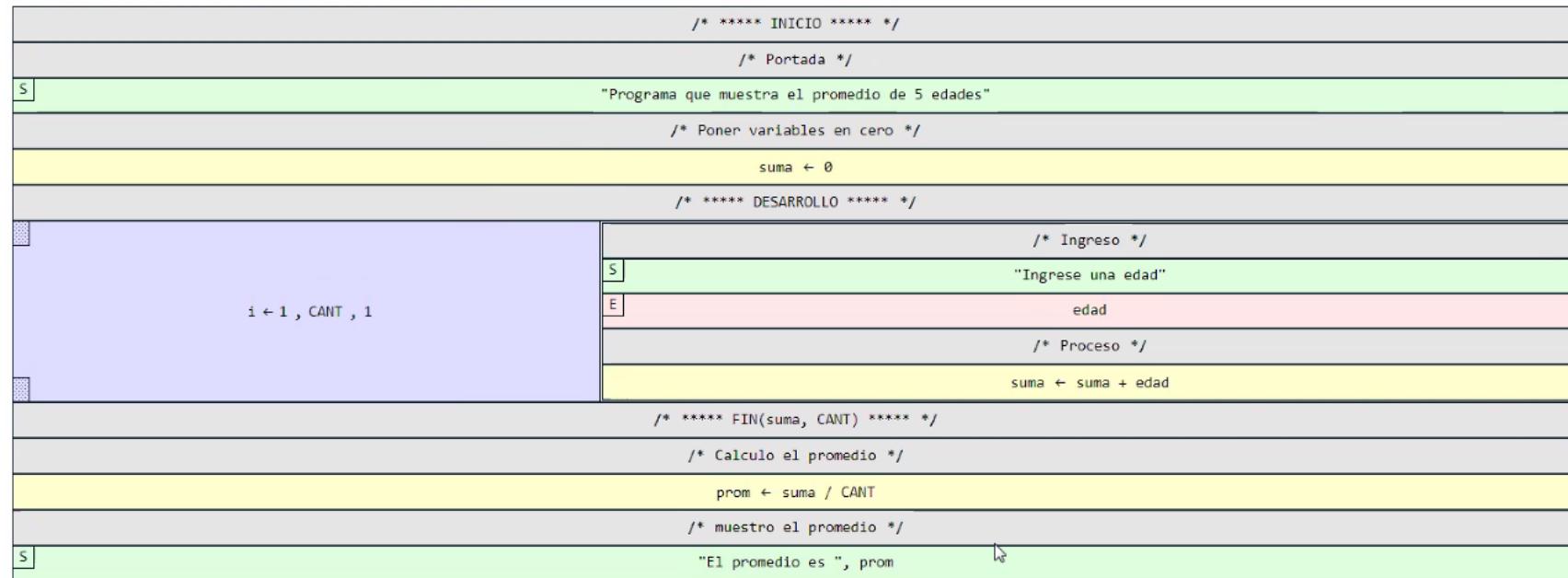
```

public void main ()
+final String FEMENINO ← "F" +final String MASCULINO ← "M" +final int EDAD_MINIMA ← 1 +final int EDAD_MAXIMA ← 120
+final int EDAD_TOPE_M ← 65 +final int EDAD_TOPE_F ← 60 +int edad +String genero

```



```
+final int CANT ← 5 +int edad, suma, i +float prom
```



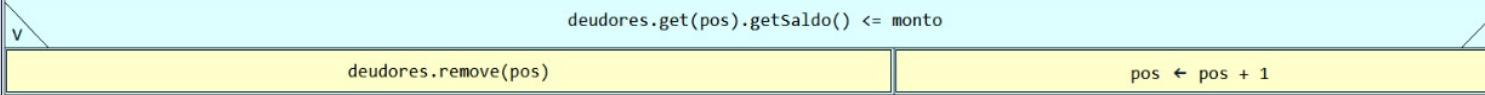
POO

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

```
+int pos ← 0
```

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

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



```
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
```

```
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
```

```
public boolean actualizarDomicilio (+ String domicilio)
+ boolean modificado = false
```

```
V domicilio!=""
```

```
setDomicilio(domicilio)
```

```
modificado = true
```

```
return modificado
```

```
class Recetario:
    public ArrayList<IngredienteAReponer> compararIngredientes (
        +ArrayList<Ingrediente> ingredientes, +double porciones, +Heladera heladera, , +Alacena alacena)
    +ArrayList<IngredienteAReponer> listaFaltantes +double cuantoNecesito
```

```
        listaFaltantes ← new ArrayList<IngredienteAReponer>()
```

```
        ingrediente.esRefrigerado()
```

```
        Ingrediente ingrediente :
            ingredientes
                V
                    cuantoNecesito ← alacena.cuantoNecesita(ingrediente.getNombre(), porciones)
```

```
                cuantoNecesito > 0
```

```
                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)
```

```
            (carreraActual.getFecha() == fecha)
```

```
                carreraDevolver ← carreraActual
```

```
                pos++
```

```
    return expresión
```

```

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 promDevolver

public ArrayList<Piloto> buscarPilotosPorDebajoDe (+ double tiempo)
+final int VUELTAS_MIN ← 10 +ArrayList<Piloto> pilotosDevolver ← new ArrayList<Piloto>()
    Piloto
    pilotoActual
    :
    this.pilotos
        (pilotoActual.dameCantVueltas() < VUELTAS_MIN) y (pilotoActual.calcularPromedio() ≤ tiempo)
            pilotosDevolver.add(pilotoActual)

        return pilotosDevolver

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

public void mostrarMenorPromedio()
+double minProm ← Double.MAX_VALUE +double promPilotoActual +Piloto pilotoMin
    Piloto
    pilotoActual :
    this.pilotos
        promPilotoActual ← pilotoActual.calcularPromedio()

        (promPilotoActual < minProm)
            minProm ← promPilotoActual
            pilotoMin ← pilotoActual

        pilotoMin

```

```

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

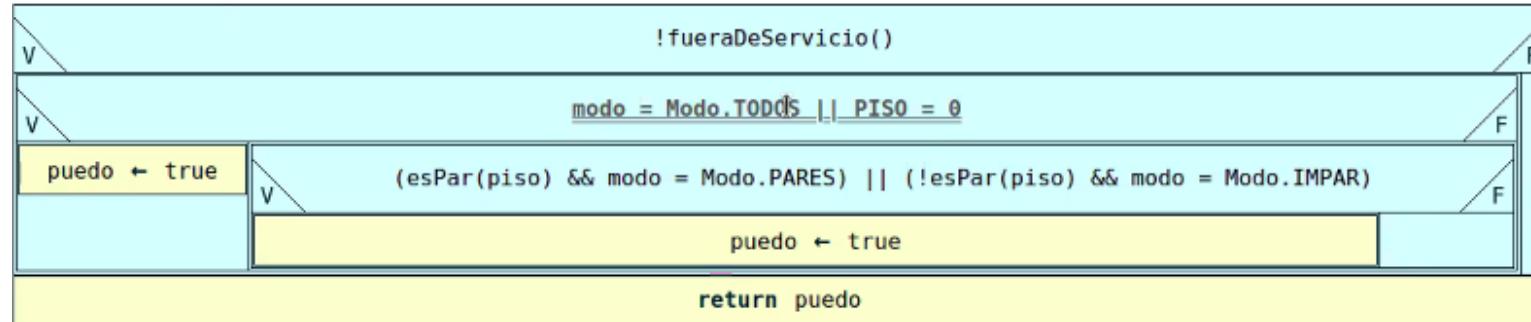
public ArrayList<Informe> pilotosPorCarrera()
{
    ArrayList<Informe> informeDevolver ← new ArrayList<Informe>()    +String fecha    +int cantPiloto
    +Informe itemInforme
    Carrera
    carreraActual :
    this.carreras
    fecha ← carreraActual.getFecha()
    cantPiloto ← carreraActual.dameCantPiloto()
    itemInforme = new Informe(fecha, cantPiloto)
    informeDevolver.add(itemInforme)
    return informeDevolver
}

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

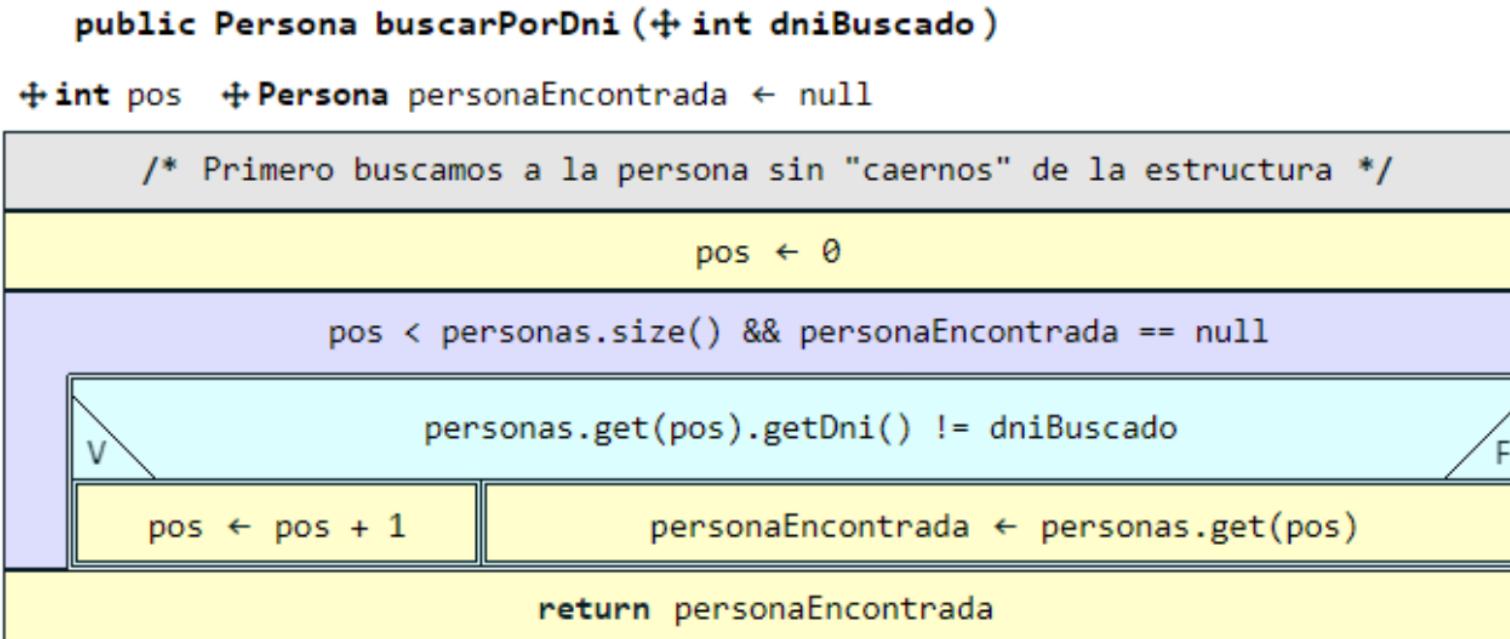
class Carrera:
    public ArrayList<Piloto> buscarPilotosPorDeabajoDe (+ double segundos)
    {
        final int CANT_VUELTAS_MIN ← 10    +ArrayList<Piloto> listaPilotos ← new ArrayList<>
        Piloto pilotoleido : pilotos
        V
        piloto.cantidadVueltas() >= CANT_VUELTAS_MIN
        F
        V
        piloto.calcularPromedio() <= segundos
        F
        listaPilotos.add(new Piloto(piloto))
        F
        listaPilotos
        return listaPilotos
    }
}

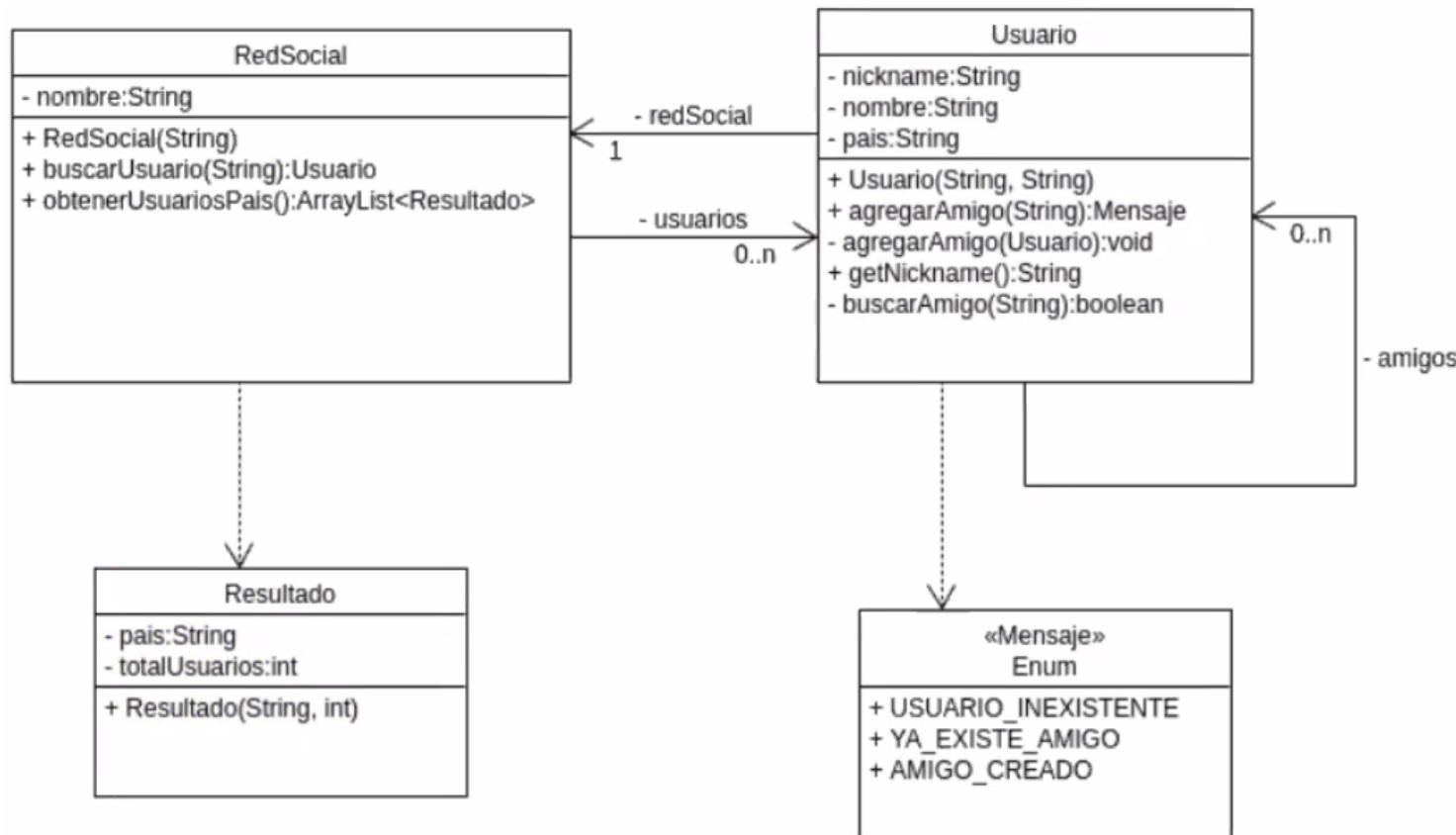
```

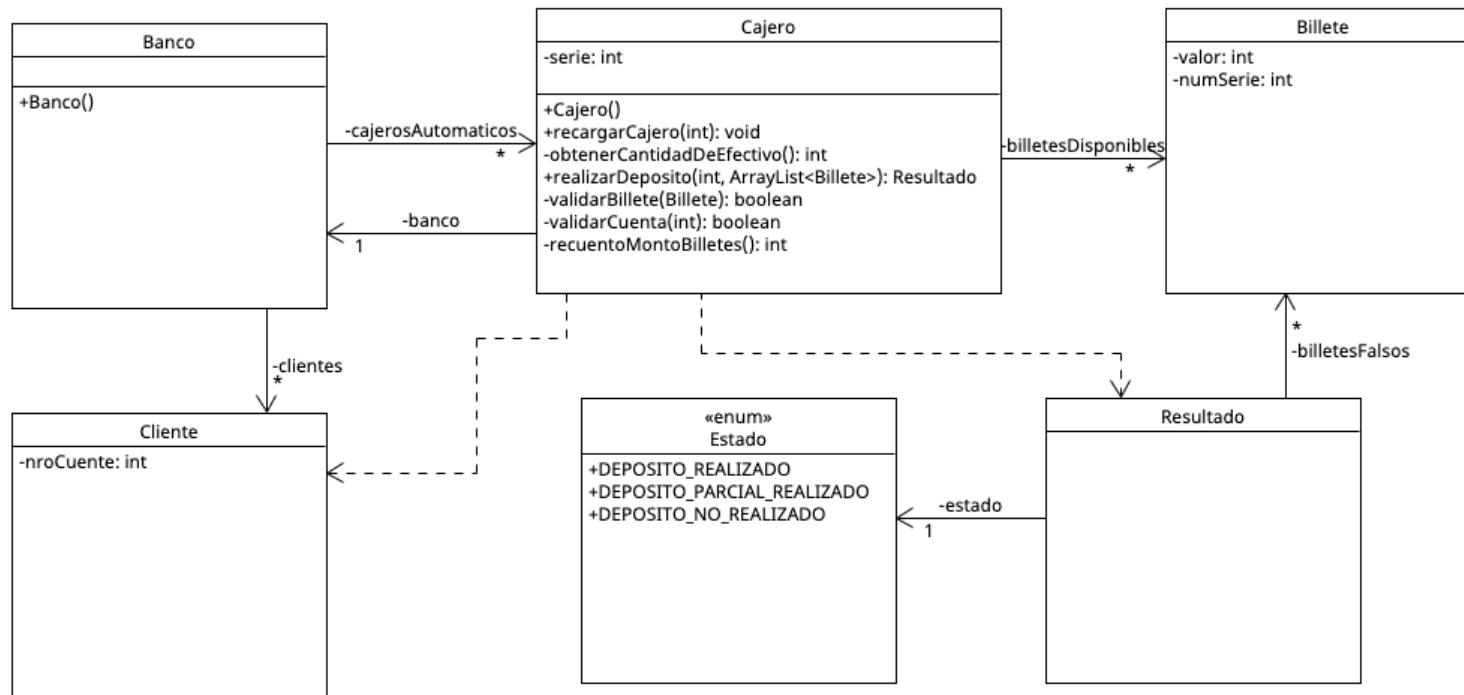
```
class Ascensor:
    public boolean puedoParar (+ int piso)
    + boolean puedo ← false
```



entender que hace es entender quien lo hace (saca la diferencia con el piso que me pidieron ejemplo piso de parámetro - piso actual)







```

class Droide:
    private ResultadoReparacion autoRepararse ()
    +int i ← 0  +Pieza pieza  +Pieza piezaOperativa  +boolean encontreAlMenosUnaPieza ← false
        i < this.piezasNoOperativas.size()
            pieza = this.piezasNoOperativas.get(i)
            piezaOperativa = this.buscarPiezaOperativa(pieza.getNombre())
            piezaOperativa != null
                encontreAlMenosUnaPieza = true
                this.reemplazarPieza(pieza, piezaOperativa)
                i++
    return this.obtenerResultadoReparacion(encontreAlMenosUnaPieza)
  
```

```

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 Autobus :
    public void dameAsientosBuenos ( + int cantNecesaria , + ArrayList<Asiento> asientosRescatados )
        +int pos ← 0 +Asiento asiento
            (pos < asientos.size()) y (cantNecesaria > 0)
            asiento ← asientos.get(pos)
            asiento.estasSano()
            asientos.remove(asiento)
            asientosRescatados.add(asiento)
            cantNecesaria--

```

The diagram illustrates the execution flow of the `dameAsientosBuenos` method. It shows a loop structure with the following steps:

- Initial state: `(pos < asientos.size()) y (cantNecesaria > 0)`
- Loop body:
 - Assign `asiento ← asientos.get(pos)`
 - Check `asiento.estasSano()`
 - Remove `asientos.remove(asiento)`
 - Add to `asientosRescatados.add(asiento)`
 - Decrement `cantNecesaria--`
- Termination: The loop exits when `pos` reaches the size of `asientos` or `cantNecesaria` reaches 0.

```
class Autobus :
    public int dameCantidadAsientosMalos ()
```

```
    +int cant ← 0
```

```
        Asiento asiento : asientos
```

```
    return cant
```

!asiento.estasSano()

cant++

```
class Autobus :
    public int dameCantidadFaltantes ()
```

```
    return cantAsientos - asientos.size()
```

```
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 Asiento :
    public boolean estasSano ()
```

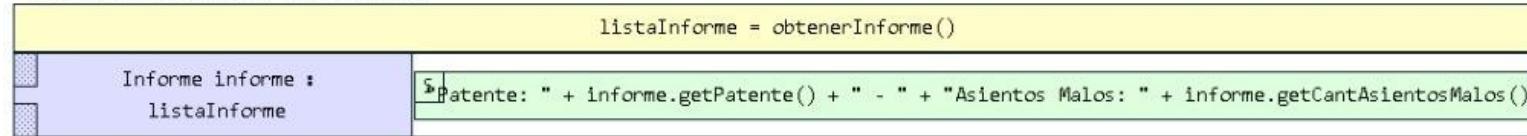
```
    return (this.estado==Estado.SANO)
```

```
class Informe :
    public Informe (+String patente , +int cantAsientosMalos )
```

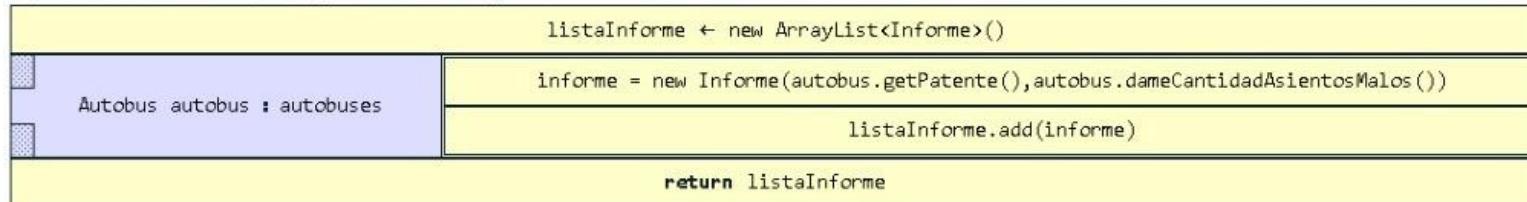
```
        this.patente ← patente
```

```
        this.cantAsientosMalos ← cantAsientosMalos
```

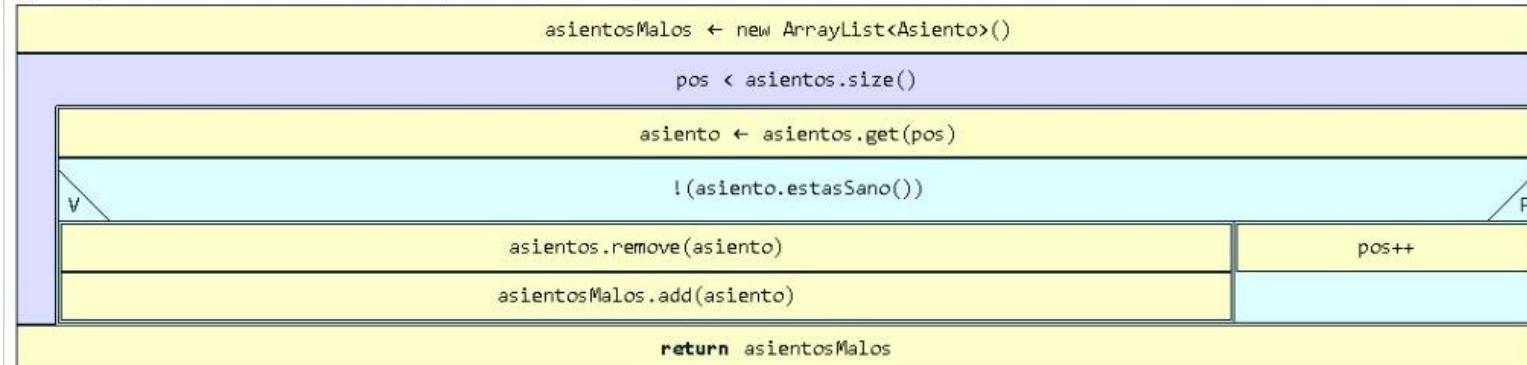
```
class Reparadora :  
    public void mostrarInforme()  
    {ArrayList<Informe>() listaInforme
```



```
    private ArrayList<Informe>() obtenerInforme()  
    { Informe informe    + ArrayList<Informe>() listaInforme
```



```
class Autobus :  
    public ArrayList<Asiento> quitarAsientosMalEstado()  
{  
    ArrayList<Asiento> asientosMalos = new ArrayList<Asiento>();  
    Asiento asiento;
```



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

```
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
    F
    return agregada
```

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

```
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 Paquete sacarPrimerPaquete()
```

```
    + Paquete paquete ← null
```

```
    V !paquetes.isEmpty()
```

```
F
```

```
    paquete ← paquetes.remove(0)
```

```
    pesoActualCarga ← pesoActualCarga - paquete.getPeso()
```

```
    return paquete
```

```
class Taller :
```

```
    public Taller( String domicilio , String telefono )
```

```
        camiones ← new ArrayList<>()
```

```
        setDomicilio(domicilio)
```

```
        setTelefono(telefono)
```

```
class Droide :
```

```
    private ArrayList<Pieza> autoRepararse()
```

```
    + int i ← 0 + Pieza pieza + Pieza piezaOperativa + ArrayList<Pieza> piezasReemplazadas ← new ArrayList<Pieza>()
```

```
    V i < this.piezas.size()
```

```
F
```

```
    pieza = this.piezas.get(i)
```

```
V
```

```
    !pieza.estaOperativa()
```

```
F
```

```
    piezaOperativa = this.buscarPiezaOperativa(pieza.getNombre())
```

```
V
```

```
    piezaOperativa != null
```

```
F
```

```
    piezasReemplazadas.add(this.reemplazarPieza(i, piezaOperativa))
```

```
i++
```

```
return piezasReemplazadas
```

```

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)
            i++
    return piezaEncontrada

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

```

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

(pos < asientos.size() && (cantNecesaria > 0))

asiento ← asientos.get(pos)

V

asiento.estasSano()

F

asientos.remove(asiento)

pos++

asientosRescatados.add(asiento)

cantNecesaria--

```
class Autobus :
    public int dameCantidadFaltantes ()
```

return cantAsientos - asientos.size()

```
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 Asiento :
    public boolean estasSano ()
```

return (this.estado==Estado.SANO)

```
class Informe :
    public Informe (⊕ String patente , ⊕ int cantFaltantes )
```

this.patente ← patente

this.cantFaltantes ← cantFaltantes

```
class Reparadora:
    public void mostrarInforme()
    +ArrayList<Informe> listaInforme
```

```
        listaInforme = obtenerInforme()
```

Informe informe :	Patente: " + informe.getPatente() + " - " + "Asientos Faltantes: " + informe.getCantFaltantes()
-------------------	-------------------------------------------------------------------------------------------------

```
class Reparadora:
    private ArrayList<Informe> obtenerInforme()
    +Informe informe +ArrayList<Informe> listaInforme
```

```
        listaInforme ← new ArrayList<Informe>()
```

Autobus autobus : autobusesRotos	informe = new Informe(autobus.getPatente(),autobus.dameCantidadFaltantes())
----------------------------------	-----------------------------------------------------------------------------

	listaInforme.add(informe)
--	---------------------------

```
    return listaInforme
```

```
class Autobus:
    public ArrayList<Asiento> quitarAsientosMalEstado()
    +ArrayList<Asiento> asientosMalos +Asiento asiento
```

```
        asientosMalos ← new ArrayList<Asiento>()
```

```
        pos < asientos.size()
```

asiento ← asientos.get(pos)	
-----------------------------	--

V	!(asiento.estasSano())	F
---	------------------------	---

asientos.remove(asiento)	pos++
--------------------------	-------

asientosMalos.add(asiento)	
----------------------------	--

```
    return asientosMalos
```

```
class Reparadora:
    public Reparadora()
```

```
        autobusesRotos ← new ArrayList<Autobus>()
```

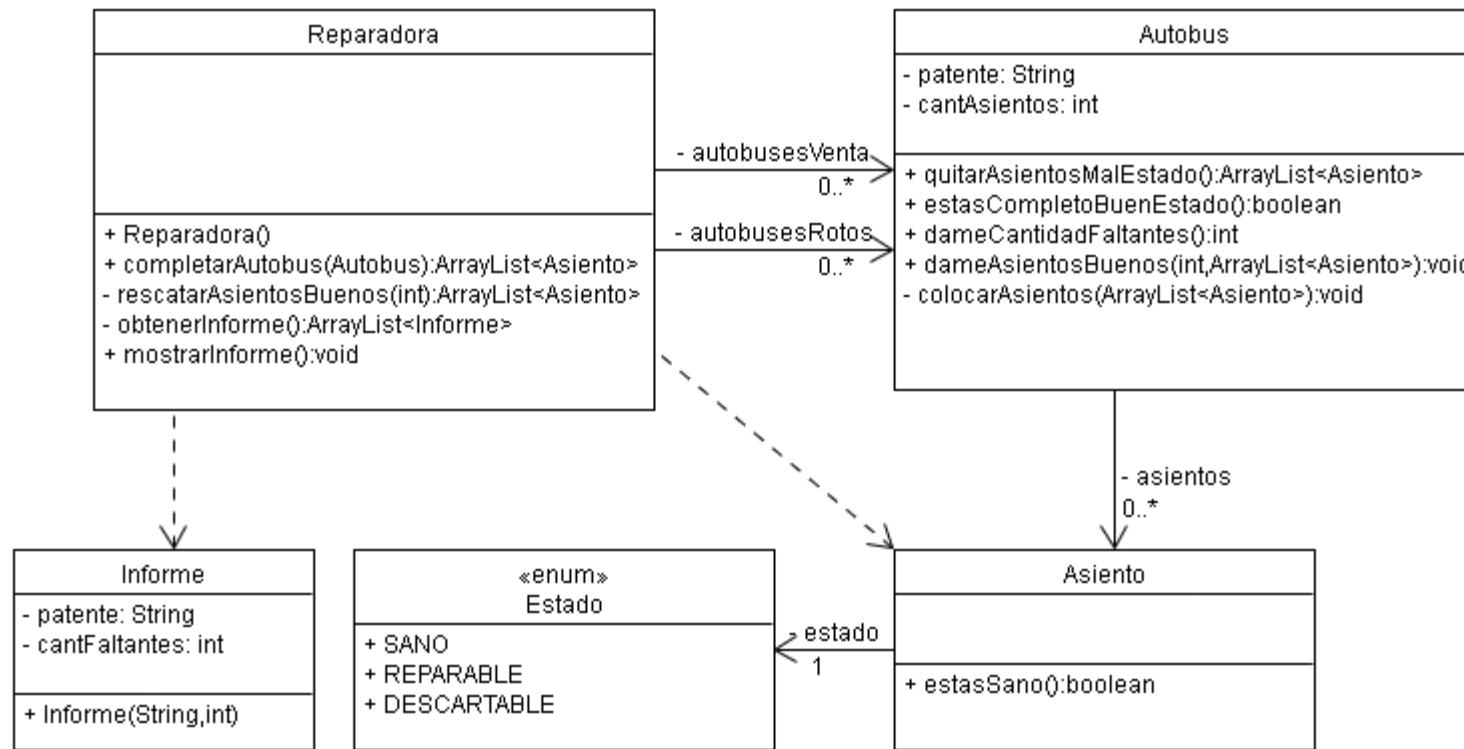
```
        autobusesVenta ← new ArrayList<Autobus>()
```

```

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

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)

```



```

class Droide:
    public Pieza buscarPiezaOperativa( String nombrePieza )
    +int i ← 0  +Droide droide  +Pieza piezaEncontrada
        i < this.droidesFueraDeServicio.size() && piezaEncontrada == null
            droide = this.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.piezasOperativas.size() && piezaEncontrada == null
```

```
            pieza = this.piezasOperativas.get(i)
```

```
            pieza.getNombre().equals(nombrePieza)
```

```
            piezaEncontrada = this.piezasOperativas.remove(i)
```

```
i++
```

```
return piezaEncontrada
```

```
class Droide:
    private ResultadoReparacion obtenerResultadoReparacion( boolean encontreAlMenosUnaPieza )
    +ResultadoReparacion resultado ← ResultadoReparacion.REPARACION_IMPOSIBLE
```

```
        this.piezasNoOperativas.isEmpty()
```

```
        resultado = ResultadoReparacion.COMPLETAMENTE_OPERATIVO
```

```
encontreAlMenosUnaPieza
```

```
        resultado = ResultadoReparacion.REPARACION_PARCIAL
```

```
return resultado
```

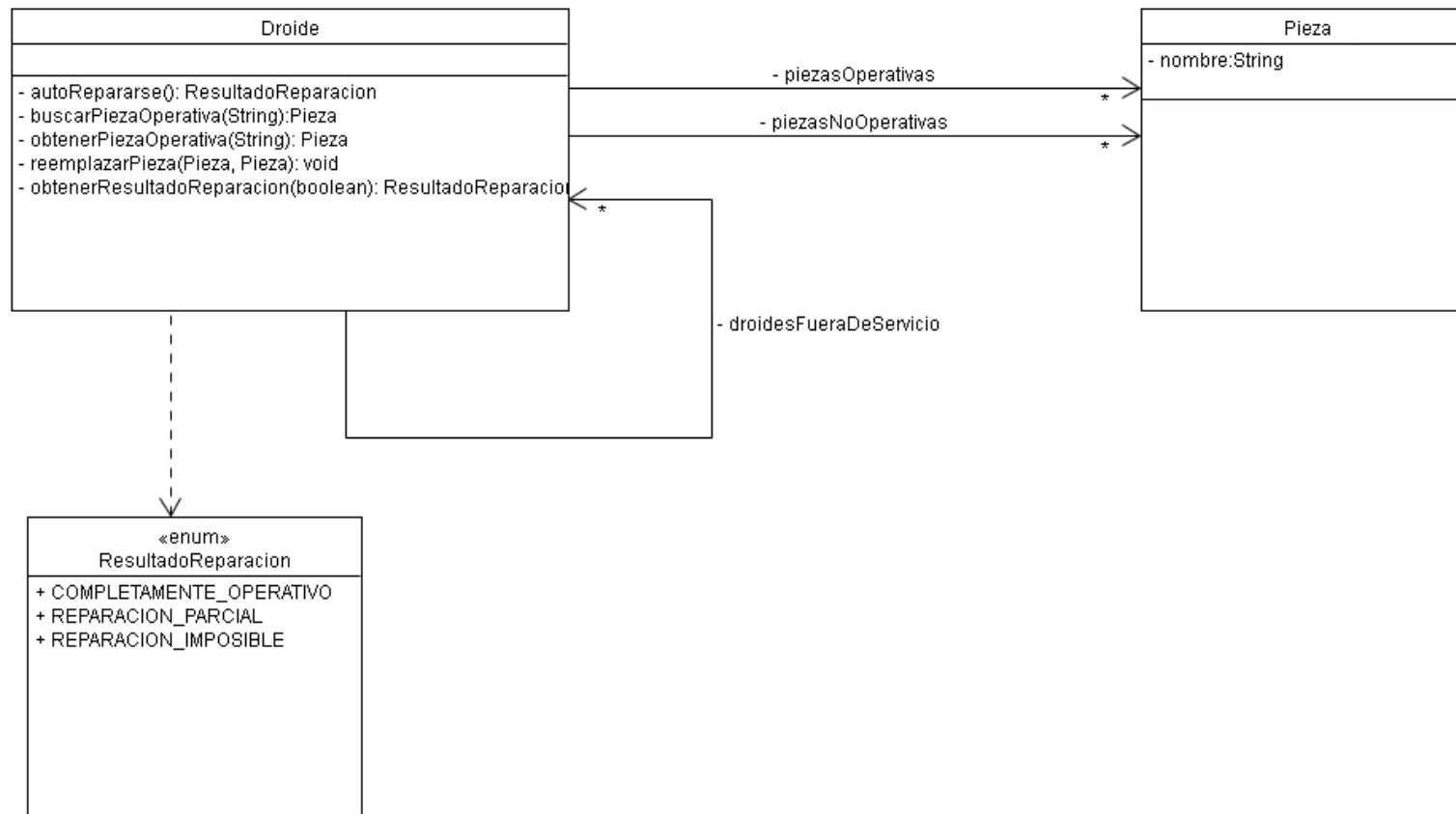
```
class Droide:
    private void reemplazarPieza( Pieza piezaNoOperativa , Pieza piezaOperativa )
```

```
        this.piezasNoOperativas.remove(piezaNoOperativa)
```

```
        this.piezasOperativas.add(piezaOperativa)
```

```
class Pieza:
    public boolean estaOperativa()
```

```
        return this.operativa
```



```

class Empresa :
    public boolean agregarPaquete( double peso , double valor , String nombreProvincia )
    boolean resultado ← false
  
```

```
    provincia ← buscarProvincia(nombreProvincia)
```

```
V (provincia != null) F
```

```
    paquetes.add(new Paquete(peso,valor,provincia))
```

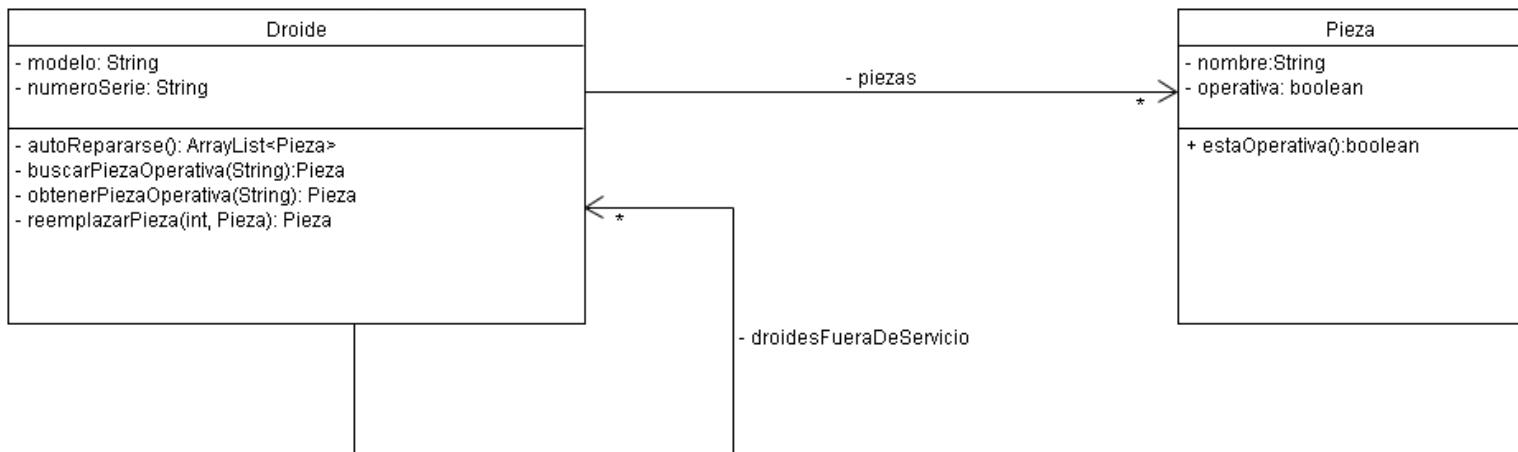
```
    resultado ← true
```

```
    return resultado
```

```
class Empresa :
```

```
    private void agregarPendiente( Paquete paquete , Motivo motivo )
```

```
    registrosPendientes.add(new RegistroPendiente(paquete,motivo))
```



```

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
}
  
```

This code snippet shows a search algorithm for a **Camion** based on its province. It initializes variables `elementoDevolver`, `pos`, `elementoActual`, and `cantElementos`. It then iterates through the list of camions, starting from index `pos` and comparing the province of each **elementoActual** to the target `provincia`. Once a match is found, it is assigned to `elementoDevolver` and the loop continues until either a match is found or the end of the list is reached.

```

class Empresa :
    private Provincia buscarProvincia( String nombre )
    +Provincia elementoDevolver ← null  +int pos ← 0  +Provincia elementoActual  +int cantElementos
        cantElementos ← provincias.size()
        (pos < cantElementos) && (elementoDevolver == null)
            elementoActual ← provincias.get(pos)
            (elementoActual.getNombre() == nombre)
                elementoDevolver ← elementoActual
                pos++
        return elementoDevolver
}

class Taller :
    public void agregarCamion( Camion camion )
        camiones.add(camion)

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

class Empresa :
    public void cargarPaquetes()
    +Camion camion  +Paquete paquete
        (paquetes.size() > 0)
        paquete ← paquetes.remove(0)
        camion ← buscarCamion(paquete.getProvincia())
        (camion != null)
            ! (camion.guardarPaquete(paquete))
                agregarPendiente(paquete,Motivo.NO_HAY_CAMION_ASIGNADO)
            agregarPendiente(paquete,Motivo.NO_HAY_LUGAR)

```

```
class Deposito {
    private void descargarCamion(↓ Camion camion)
    ↓ Paquete paquete
```

```
        paquete ← camion.sacarPrimerPaquete()
        paquete != null
        paquetes.add(paquete)
        paquete ← camion.sacarPrimerPaquete()
```

```
class EmpresaDeTransporte {
    public EmpresaDeTransporte(↓ String nombre)
```

```
        setNombre(nombre)
        depositos ← new ArrayList<>()
```

```
class Camion {
    public double getPorcentajeDeCarga()
```

```
        return pesoActualCarga * 100 / TOPE_PESO
```

```
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 EmpresaDeTransporte {
    private ArrayList<String> obtenerDepositosSegunCriteria(↓ int cantidadDeCamiones, ↓ double porcentajeMinimo)
    ↓ ArrayList<String> provincias
```

```
        provincias ← new ArrayList<>()
```

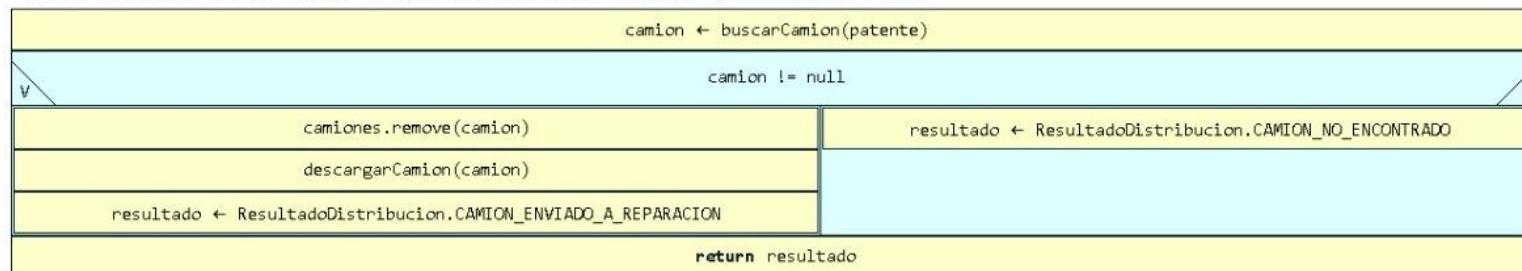
```
        Deposito deposito : depositos
```

```
        ↓ V deposito.tieneCamionesSegunCriteria(cantidadDeCamiones, porcentajeMinimo)
```

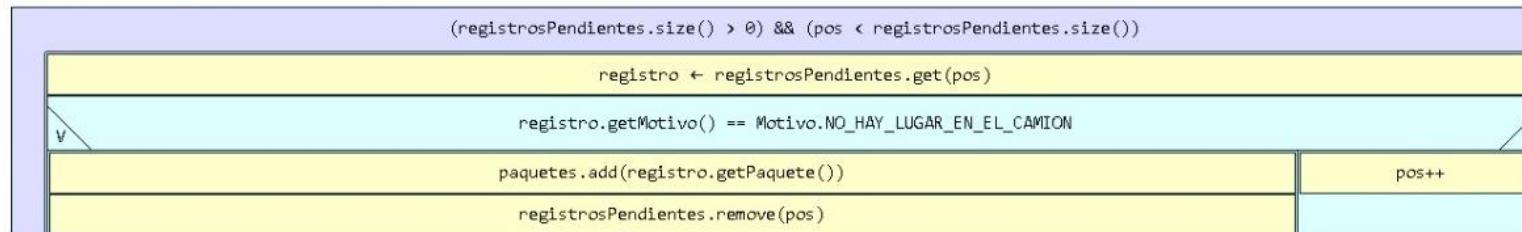
```
        ↓ F provincias.add(deposito.getProvincia())
```

```
        return provincias
```

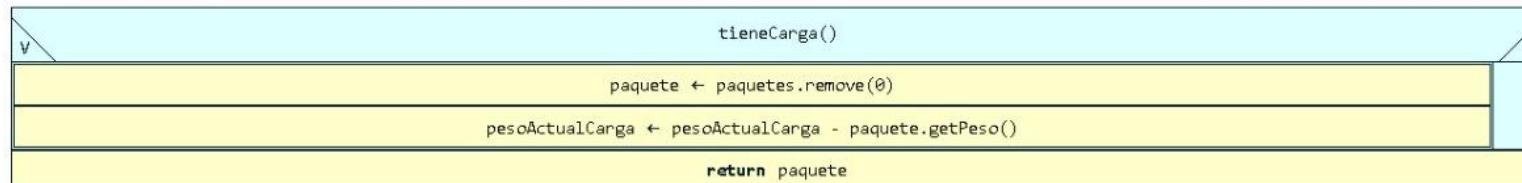
```
class Deposito :
    private RESULTADO_DISTRIBUCION quitarCamionPorDesperfecto( String patente )
    + Paquete paquete + Camion camion + ResultadoDistribucion resultado + boolean pudo ← true
```



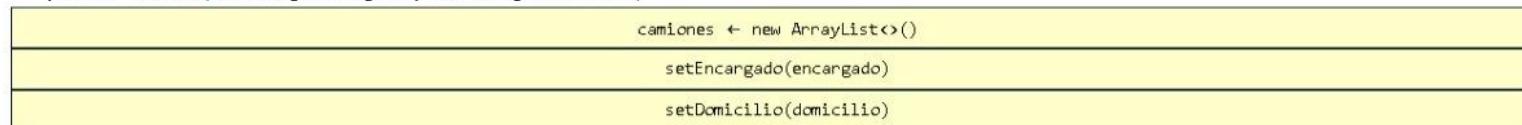
```
class Empresa :
    public void reincorporarPaquetes()
    + int pos ← 0 + RegistroPendiente registro
```

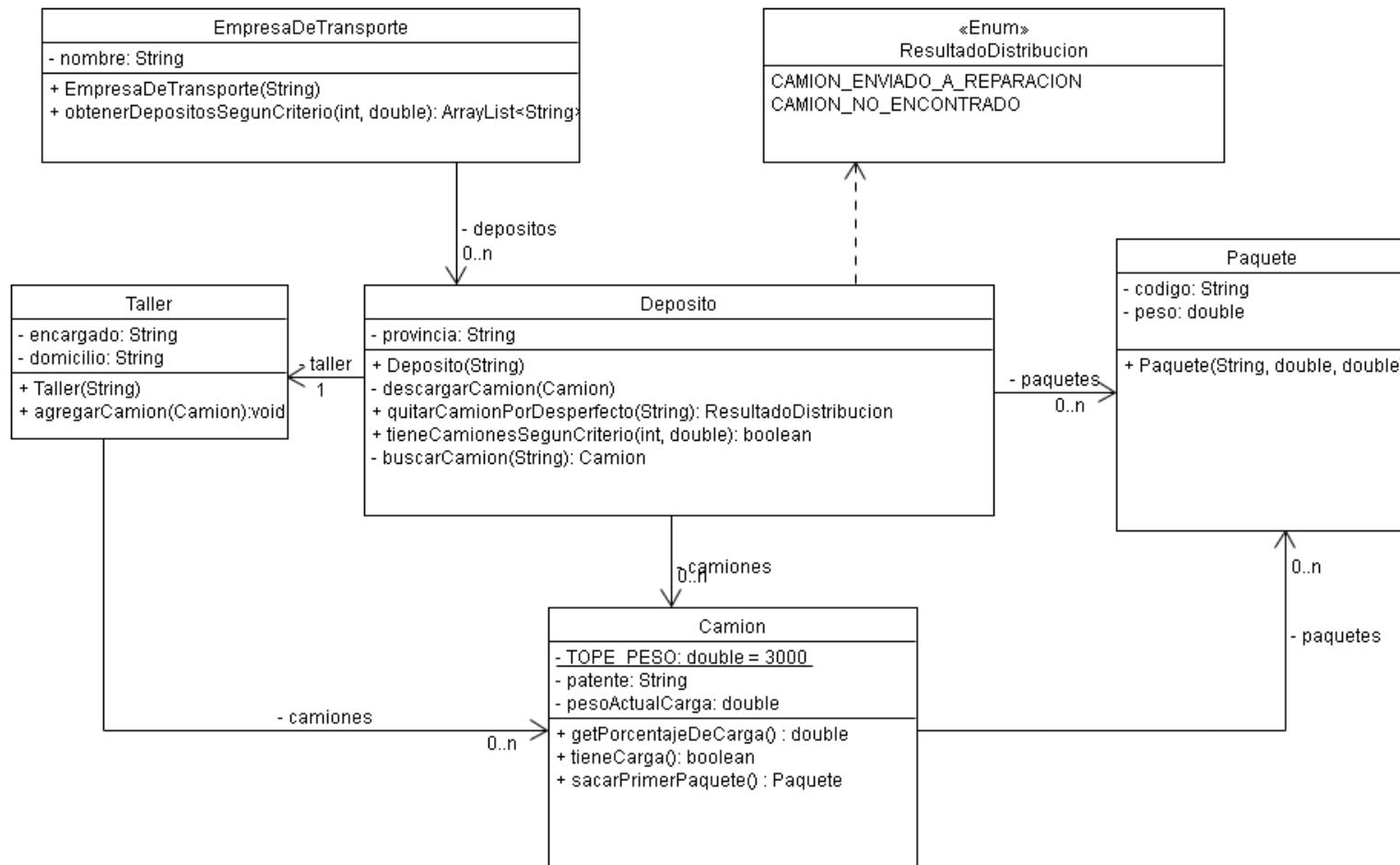


```
class Camion :
    public Paquete sacarPrimerPaquete()
    + Paquete paquete ← null
```



```
class Taller :
    public Taller( String encargado , String domicilio )
```





```

class Deposito :
    public boolean tieneCamionesSegunCriteria( int topeCamiones , int tipo porcRequerido )
    +int cantCamiones ← 0  +int pos ← 0
  
```

```
    pos < camiones.size() && cantCamiones < topeCamiones
```

```
    camiones.get(pos).getPorcentajeDeCarga() >= porcRequerido
```

```
    cantCamiones++
```

```
    pos++
```

```
    return cantCamiones == topeCamiones
```

```

class Camion :
    public boolean tieneCarga ()
        return !paquetes.isEmpty()

class MaxMesa :
    public void agregarNumero (int numeroMesa)
        this.numeros.add(numeroMesa)

class Padron :
    private boolean agregarPersona (Persona persona)
    +boolean pude ← false
    V
        this.buscarEscuelaPorCodigoPostal(persona.obtenerCodigoPostal()) != null
    F
        this.personas.add(persona)
        pude = true
    return pude

class Padron :
    public void agregarVotantes (ArrayList<Persona> personas)
    +int i ← 0 +Persona persona
    V
        i < personas.size()
        persona = personas.get(i)
        this.agregarPersona(persona)
    F
        personas.remove(i)
        i ++
    return

class Deposito :
    public Camion buscarCamion (String patente)
    +Camion camion +int pos
    V
        camion ← null
        pos ← 0
        pos < camiones.size() && camion == null
    F
        camion.get(pos).getPatente().equals(patente)
    F
        camion ← camion.get(pos)
        pos++
    return camion

```

```
class Padron:
    private Escuela buscarEscuela( String nombre )
```

```
    +Escuela escuela +Escuela escuelaEncontrada ← null +int i ← 0
```

```
        i < this.escuelas.size() && escuelaEncontrada == null
```

```
            escuela = this.escuelas.get(i)
```

```
            escuela.getNombre().equals(nombre)
```

V

F

```
            escuelaEncontrada = escuela
```

```
i ++
```

```
    return escuelaEncontrada
```

```
class Padron:
    private Escuela buscarEscuelaPorCodigoPostal( String codigoPostal )
```

```
    +Escuela escuela +Escuela escuelaEncontrada ← null +int i ← 0
```

```
        i < this.escuelas.size() && escuelaEncontrada == null
```

```
            escuela = this.escuelas.get(i)
```

```
            escuela.obtenerCodigoPostal().equals(codigoPostal)
```

V

F

```
            escuelaEncontrada = escuela
```

```
i ++
```

```
    return escuelaEncontrada
```

```
class Camion:
    public boolean cargarPaquete( Paquete paquete )
```

```
    +double cargaRealizada ← false
```

```
        hayLugar(paquete.getPeso())
```

```
            paquetes.add(paquete)
```

```
            pesoActualCarga ← pesoActualCarga + paquete.getPeso()
```

```
            cargaRealizada ← true
```

```
    return cargaRealizada
```

V

F

```
class Deposito :
    private boolean cargarPaquete( Paquete paquete )
    +int pos
    +boolean paqueteCargado
```

```
        paqueteCargado ← false
```

```
        pos ← 0
```

```
        pos < camiones.size() && !paqueteCargado
```



```
    return paqueteCargado
```

```
class Escuela :
    public void designarPresidenteDeMesa( Mesa mesa , Persona persona )
```

```
        this.quitarPersonaDeMesa(persona)
```

```
        mesa.asignarPresidente(persona)
```

```
class Deposito :
    private void distribuirCarga( Camion camion )
    +Paquete paquete
```

```
        camion.ordenarCarga()
```

```
        paquete ← camion.sacarPrimerPaquete()
```

```
        paquete != null
```

```
V --> F
    F : !cargarPaquete(paquete)
```

```
    paquetes.add(paquete)
```

```
    paquete ← camion.sacarPrimerPaquete()
```

```
class Padron :
```

```
    private ArrayList<PadronEscuela> generarPadron( String nombreEscuela , ArrayList<Persona> personasEnEscuela )
```

```
    +ArrayList<PadronEscuela> padronEscuela ← new ArrayList<PadronEscuela>()
```

```
    Persona persona :
    personasEnEscuela
```

```
        padronEscuela.add(new PadronEscuela(nombreEscuela, persona.getDni(), persona.getNombreApellido()))
```

```
    return padronEscuela
```

```
class Persona :
```

```
    public String getNombreApellido()
```

```
        return this.nombre + " " + this.apellido
```

```

class Persona:
    public String getNombreApellido()
        return this.nombre + " " + this.apellido

class Camion:
    private boolean hayLugar(† double pesoPaquete)
        return (pesoPaquete <= TOPE_PESO - pesoActualCarga)

class Padron:
    public ArrayList<PadronEscuela> obtenerInforme(† String nombreEscuela)
        †ArrayList<Personas> personasEnEscuela † Escuela escuela †ArrayList<PadronEscuela> padronEscuela
            escuela = this.buscarEscuela(nombreEscuela)
            escuela != null
            personasEnEscuela = this.obtenerPersonasPorCodigoPostal(escuela.obtenerCodigoPostal())
            padronEscuela = this.generarPadron(escuela.getNombre(), personasEnEscuela)
        return padronEscuela

class Mesa:
    public void obtenerPadronMesa(† ArrayList<PadronMesa> padronMesas )
        † int orden ← 0
        Persona persona : this.personas
            orden ++
            padronMesas.add(new PadronMesa(this.numero, orden, persona.getDni(), persona.getNombreApellido()))

class Padron:
    private ArrayList<Persona> obtenerPersonasPorCodigoPostal(† String codigoPostal)
        †ArrayList<Persona> personas ← new ArrayList<Personas>()
        Persona persona : this.personas
            V persona.obtenerCodigoPostal().equals(codigoPostal)
            F personas.add(persona)
        return personas

class PadronEscuela:
    public PadronEscuela(† String nombreEscuela, † String dni, † String nombreApellido)
        this.nombreEscuela = nombreEscuela
        this.dni = dni
        this.nombreApellido = nombreApellido

```

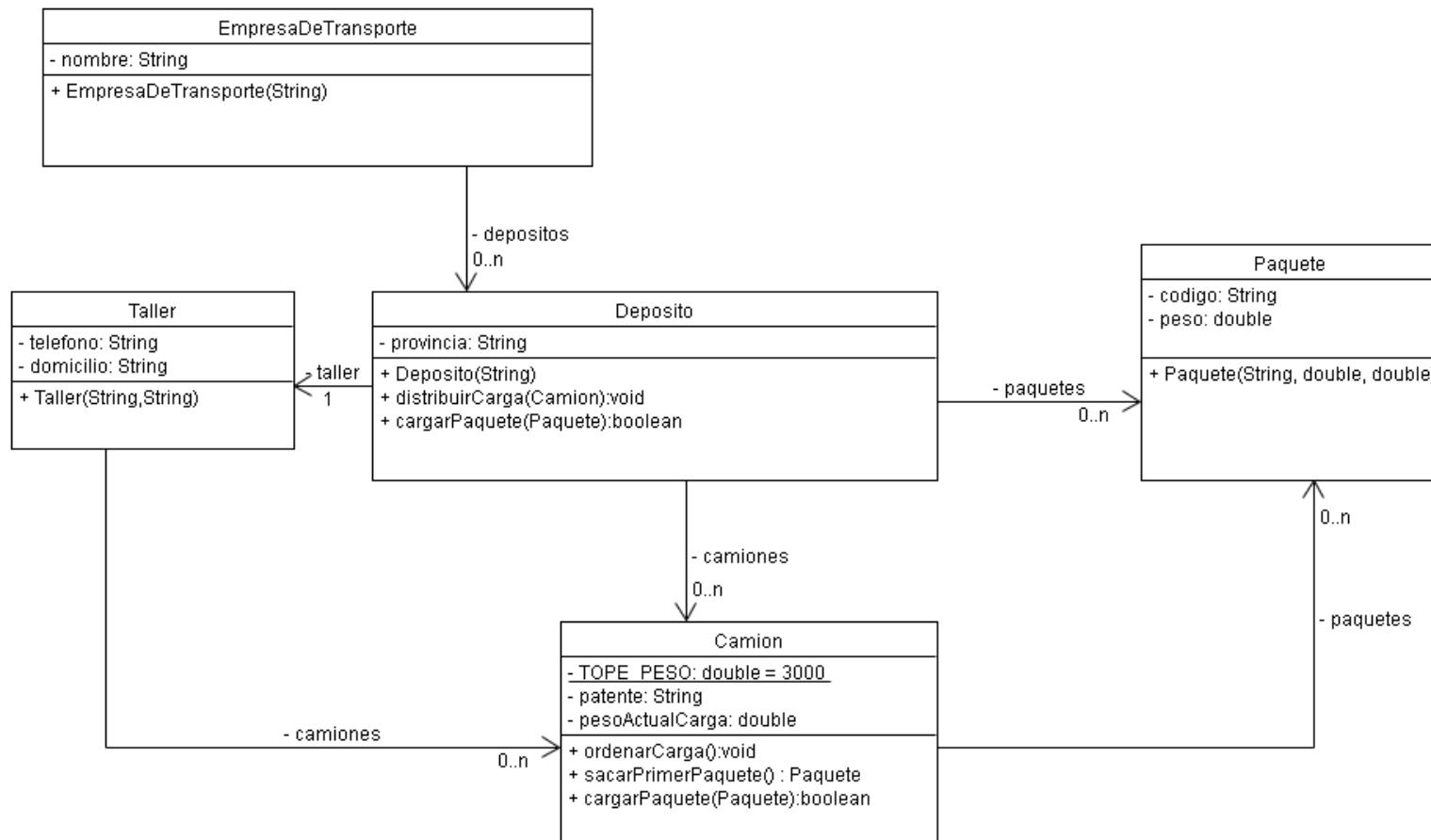
```
class PadronMesa :
    public PadronMesa(⊕ int numeroMesa , ⊕ int orden , ⊕ String dni , ⊕ String nombreApellido)
        this.numeroMesa = numeroMesa
        this.orden = orden
        this.dni = dni
        this.nombreApellido = nombreApellido
```

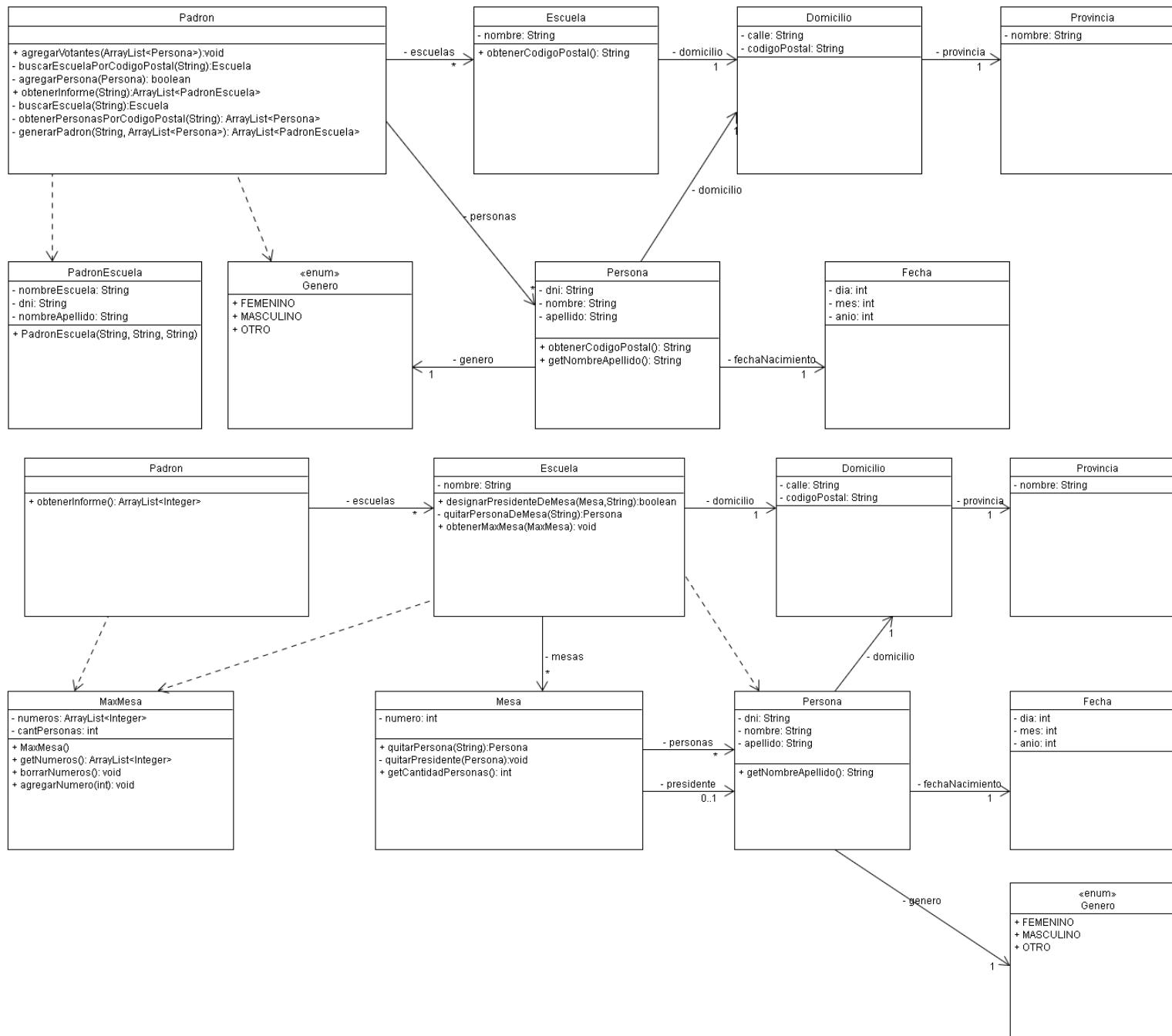
```
class Mesa :
    public boolean quitarPersona(⊕ Persona persona)
```

```
⊕ int i ← 0   ⊕ Persona personaQuitada ← null   ⊕ Persona personaActual
    i < this.personas.size() && personaQuitada == null
    personaActual = this.personas.get(i)
    personaActual == persona
    personaQuitada = this.personas.remove(i)
    this.quitarPresidente(persona)
    i ++
    return (personaQuitada != null)
```

```
class Escuela :
    private void quitarPersonaDeMesa(⊕ Persona persona)
```

```
⊕ int i ← 0   ⊕ boolean personaQuitada ← false   ⊕ Mesa mesa
    i < this.mesas.size() && !personaQuitada
    mesa = this.mesas.get(i)
    personaQuitada = mesa.quitarPersona(persona)
    i ++
```

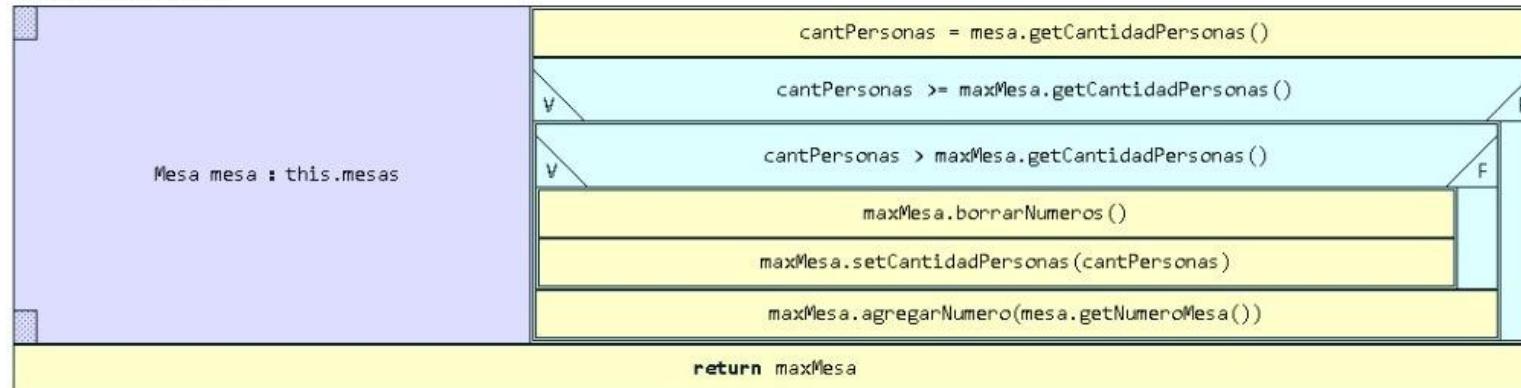




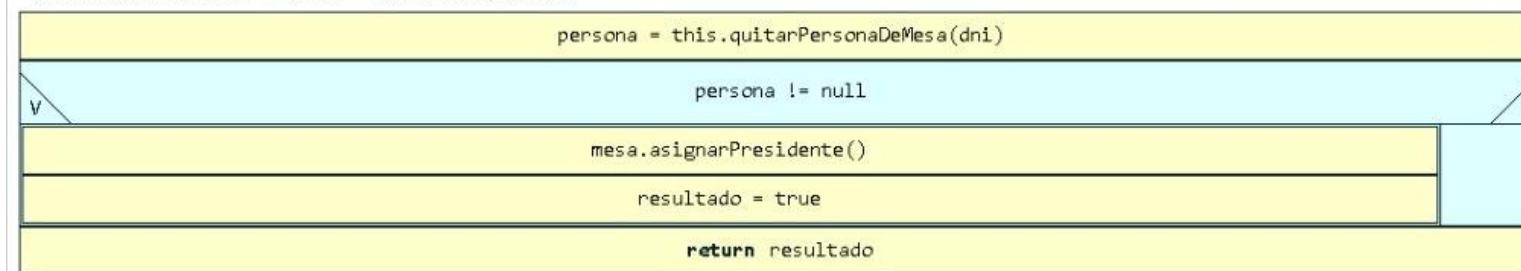
```
class Mesa :  
    public void asignarPresidente(† Persona persona)  
        this.presidente = persona  
        this.personas.add(persona)  
  
class Mesa :  
    public int getCantidadPersonas ()  
        return this.personas.size()  
  
class Persona :  
    public String obtenerCodigoPostal ()  
        return this.domicilio.getCodigoPostal()  
  
class Escuela :  
    public ArrayList<PadronMesa> obtenerInforme ()  
    †ArrayList<PadronMesa> padronMesas ← new ArrayList<PadronMesa>  
    Mesa mesa : this.mesas  
    mesa.obtenerPadronMesa(padronMesas)  
    return padronMesas  
  
class Mesa :  
    public void asignarPresidente(† Persona persona)  
        this.presidente = persona  
        this.personas.add(persona)  
  
class Mesa :  
    public int getCantidadPersonas ()  
        return this.personas.size()  
  
class Persona :  
    public String obtenerCodigoPostal ()  
        return this.domicilio.getCodigoPostal()  
  
class MaxMesa :  
    public void borrarNumeros ()  
        this.numeros.clear()  
  
class Escuela :  
    public String obtenerCodigoPostal ()  
        return this.domicilio.getCodigoPostal()
```

```
class Mesa :  
    private void quitarPresidente(† Persona persona )  
        this.presidente == persona  
        V  
        this.presidente = null  
        F  
  
class MaxMesa :  
    public void borrarNumeros ()  
        this.numeros.clear()  
  
class Escuela :  
    public String obtenerCodigoPostal ()  
        return this.domicilio.getCodigoPostal()  
  
class Mesa :  
    private void quitarPresidente(† Persona persona )  
        this.presidente == persona  
        V  
        this.presidente = null  
        F  
  
class MaxMesa :  
    public ArrayList<Integer> getNumeros ()  
        return this.numeros  
  
class MaxMesa :  
    public MaxMesa ()  
        this.numeros = new ArrayList<Integer>()  
        this.cantidadPersonas = 0
```

```
class Escuela:
    public void obtenerMaxMesa (⊕ MaxMesa maxMesa)
    +int cantPersonas
```



```
class Escuela:
    public boolean designarPresidenteDeMesa (⊕ Mesa mesa , ⊕ String dni)
    +boolean resultado ← false  +Persona persona
```



```
class Padron:
    public ArrayList<Integer> obtenerInforme ()
    +MaxMesa maxMesa  +MaxMesa maxMesaEscuela
```



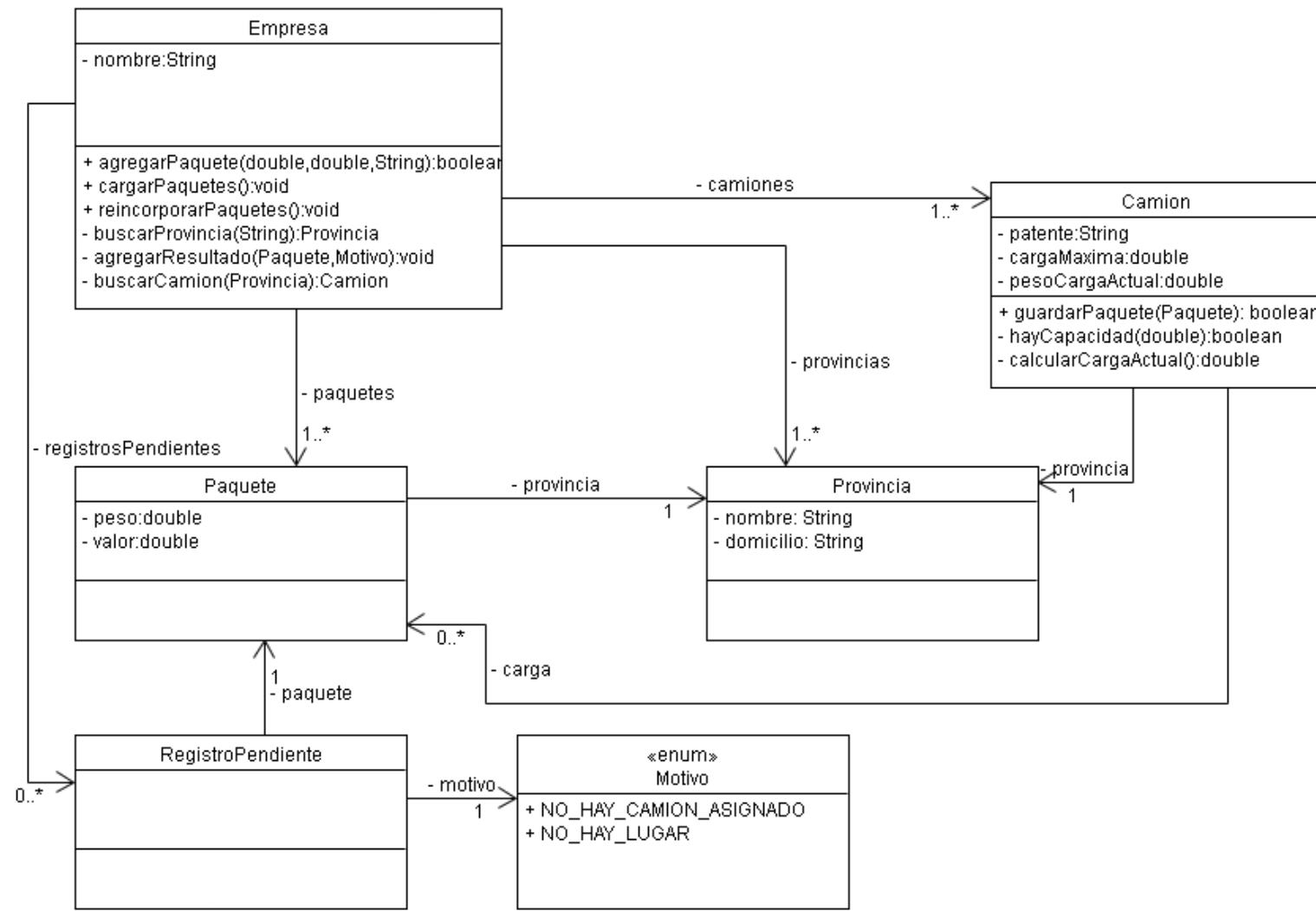
```

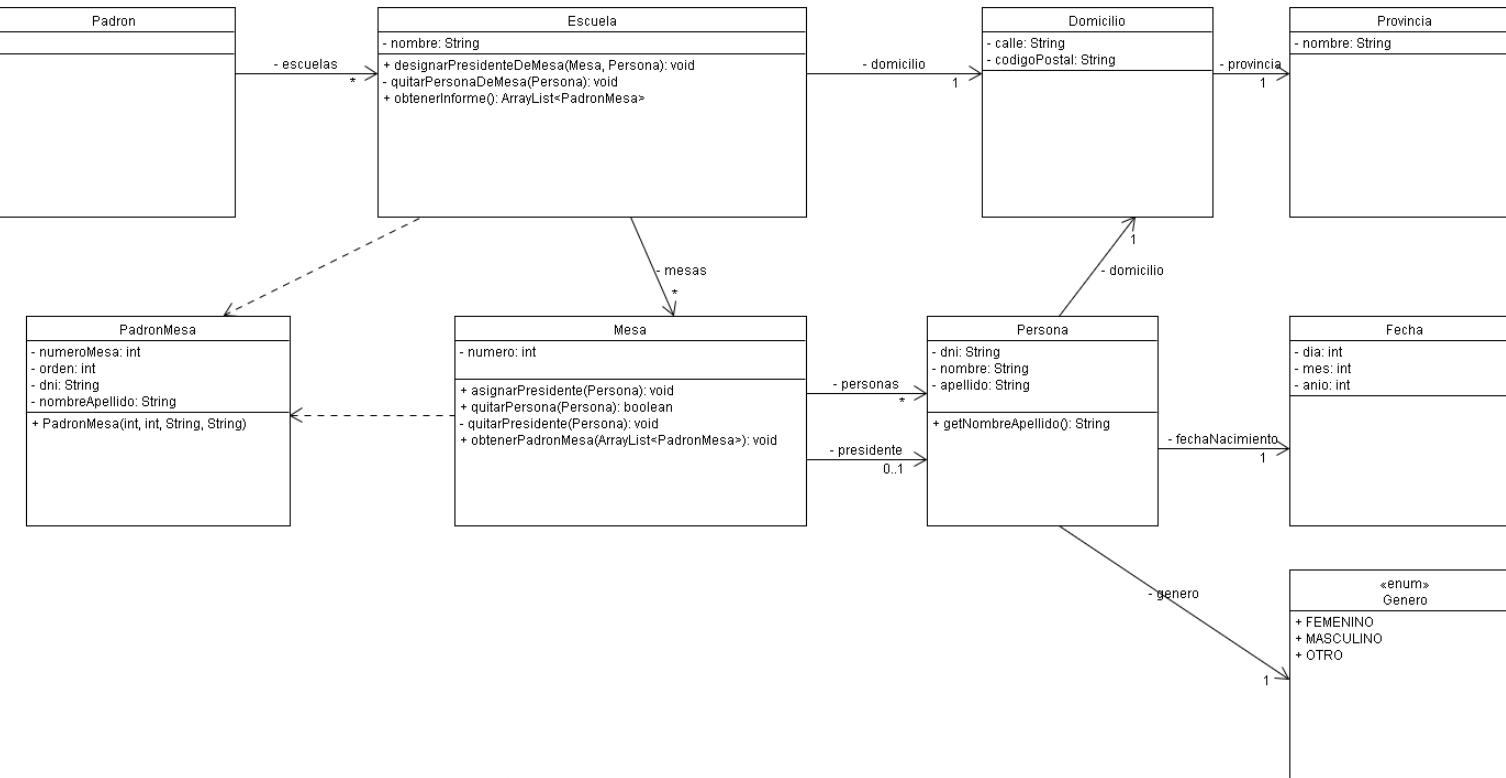
class Mesa:
    public Persona quitarPersona( String dni )
    +int i ← 0  +Persona personaQuitada ← null  + Persona persona
        i < this.personas.size() && personaQuitada == null
            persona = this.personas.get(i)
            persona.getDni().equals(dni)
            personaQuitada = this.personas.remove(i)
            i++
            this.quitarPresidente(persona)
        return personaQuitada
    }

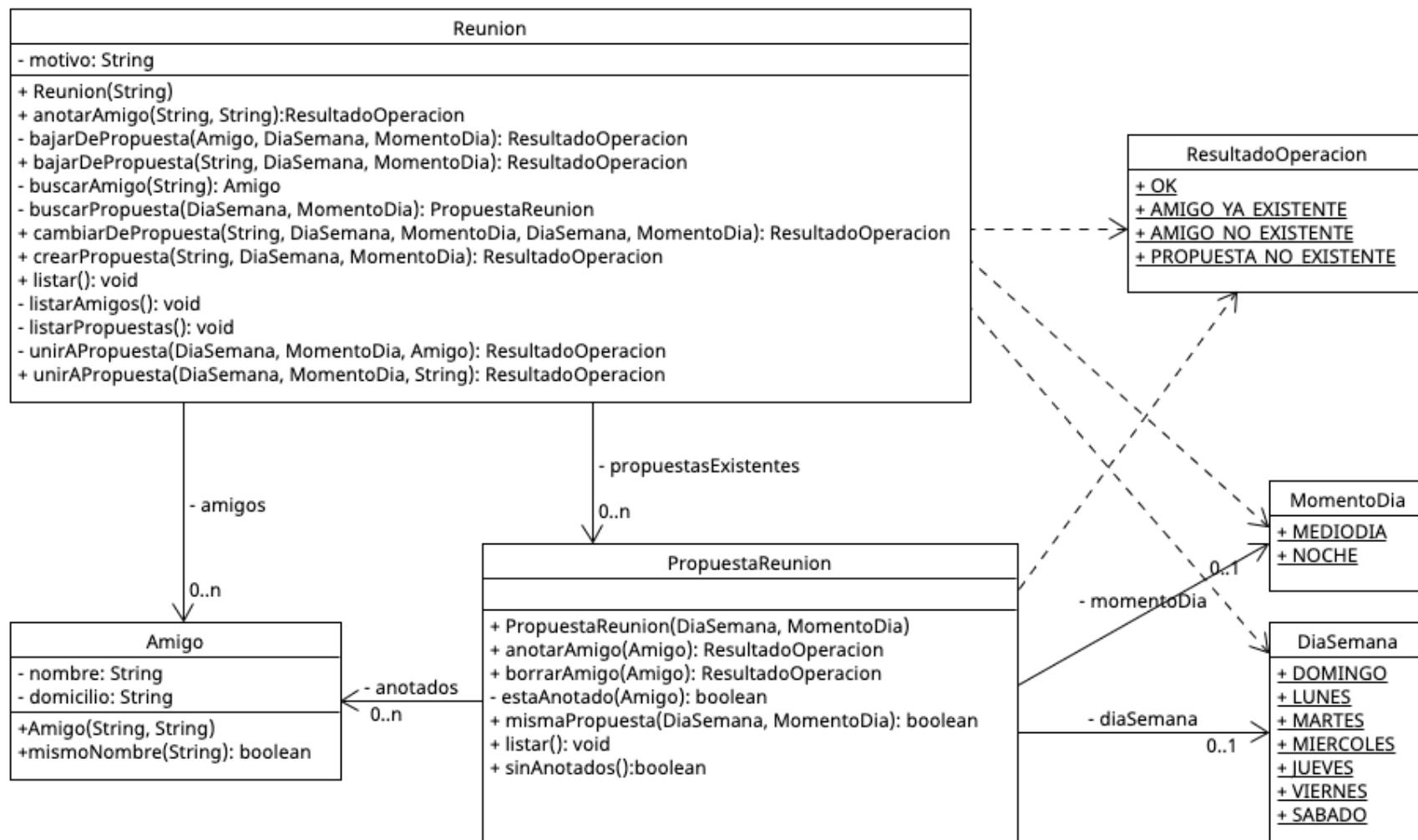
class Escuela:
    private Persona quitarPersonaDeMesa( String dni )
    +Persona personaQuitada ← null  +int i ← 0  +Mesa mesa
        i < this.mesas.size() && !personaQuitada
            mesa = this.mesas.get(i)
            personaQuitada = mesa.quitarPersona(dni)
            i ++
        return personaQuitada
    }

class Mesa:
    private void quitarPresidente( Persona persona )
        this.presidente == persona
        this.presidente = null
    }

```







La empresa se dedica al transporte ferroviario y nos pide un sistema para organizar los armados de las distintas formaciones de trenes.

Todos los trenes **salen de la terminal situada en Avellaneda 1242** y de cada tren conocemos cuál es su destino de llegada (String) y el maquinista que posiblemente tenga asignado.

La empresa realiza la maniobra para armar las distintas formaciones y dejarlas en condiciones de ser operadas por un maquinista. Para ello tiene una colección de vagones libres y otra de trenes ya conformados.

De cada vagón conocemos su número de serie (String), un tipo de vagón (locomotora, carga o pasajero) y un estado booleano que indica si está en reparación o no.

La empresa lleva un registro con todos los maquinistas que tiene contratados. De cada maquinista se conoce su nombre, número de documento y el tren al que está asignado (o null si no estuviera asignado a ningún tren por el momento).

Un tren se conforma con una locomotora que encabeza la formación, más una cantidad de vagones de carga y/o una cantidad de vagones de pasajeros.

Se considera que una formación es clásica cuando está compuesta exactamente por una locomotora más 8 vagones de pasajeros.

Nota: Tener en cuenta que en todos los casos cuando se menciona locomotora, la misma es un Vagón que su atributo "tipo de vagón" es LOCOMOTORA

Enunciado final diciembre 2021

"Luz verde, luz roja"

Una organización clandestina lleva a cabo una competencia donde los jugadores arriesgan su vida a cambio de una importante suma de dinero para el único ganador/superviviente.

Nos piden simular el juego conocido como "Luz verde, luz roja".

Este juego consiste en cruzar la meta, la cual se encuentra a 150 metros de donde se encuentran los jugadores al empezar. Todos los jugadores están en un extremo (el metro cero). En el otro extremo de la pista (donde los jugadores deben llegar) se encuentra una muñeca gigante. Si la muñeca les da la espalda los jugadores tienen "luz verde" y pueden correr hacia la meta y ponerse a salvo. Pero cuando la muñeca se dé vuelta para enfrentarlos se decretará la "luz roja" y todos los jugadores deberán detenerse y quedarse "congelados". Todo aquel jugador que se detecte en movimiento será eliminado.

La muñeca completará un total de 10 rondas cambiando de estado (una luz verde y una roja por ronda): primero "luz verde", cuando los jugadores serán libres de moverse y correr todos los metros que puedan. Cuando la muñeca gire y se prenda la luz roja, la muñeca los "observará", revisando uno a uno a cada jugador para detectar quién está en movimiento. Quien se esté moviendo será eliminado.

En cambio, cuando un jugador alcance o supere la meta estará a salvo y deberá moverse junto a los otros supervivientes. También se moverán (a otro lado) los que sean eliminados. Al cumplirse la décima luz verde y prenderse la luz roja por última vez todos aquellos jugadores que no hayan conseguido llegar a la meta serán eliminados sin importar los metros recorridos.

Cada jugador es identificado de forma única con un número entero. Si hiciese falta se pueden agregar los atributos que considere necesarios para resolver el comportamiento descrito antes.

Métodos provistos:

Para saber cuántos metros avanzó un jugador en una ronda se provee el método:

`private double obtenerDistanciaRecorrida()` de la clase Jugador, el cual no recibe parámetros y siempre devuelve un número que representa la cantidad de metros que el jugador se desplazó en una ronda.

Para saber si un jugador está quieto durante una luz roja se provee el método:

`public boolean estaEnMovimiento()` de la clase Jugador, el cual no recibe parámetros y devuelve un valor booleano indicando si el jugador está moviéndose o no.

Estos dos métodos provistos NO deben diagramarse en NS+, solamente deben agregarse al diseño e invocarse en donde se considere necesario al desarrollar los métodos requeridos en NS+.

Se pide:

Confeccionar el diagrama UML que describe el escenario del enunciado incluyendo los atributos de cada clase y los métodos a desarrollar y aquellos que creas conveniente.

Desarrollar el método `jugar()` de la clase Juego que no recibe parámetros y devuelve uno de los siguientes valores:

`UNICO_GANADOR`: cuando sobrevive exactamente un solo jugador.

`MENOS_DE_LA_MITAD`: cuando la cantidad de jugadores que sobrevivieron al juego sea menor o igual a la mitad de los jugadores que arrancaron a jugar.

`MAS_DE_LA_MITAD`: cuando la cantidad de jugadores que sobrevivieron al juego sea mayor a la mitad de los jugadores que arrancaron a jugar.

`SIN_SOBRVIVIENTES`: cuando ningún jugador haya sobrevivido.

Nota 1: Para desarrollar el juego, de cada jugador también se deben saber el número de ronda en el que llegó a la meta ó en la que fue eliminado, y la cantidad de metros recorridos.

Nota 2: En el desarrollo de este método se dará especial énfasis a la correcta modularización y distribución de tareas y/o responsabilidades.

Importante: El juego posee una colección con todos los jugadores que participarán del mismo y dos colecciones más donde quedarán distribuidos los jugadores al terminar el juego:

En una colección deben quedar cargados aquellos jugadores que sobrevivieron.

En otra colección deben quedar cargados los jugadores que fueron eliminados.

La colección original debe quedar vacía.

Enunciado recu dic2021

Una compañía discográfica llamada “The Chemicals”, se especializa en la edición de discos de algunos géneros muy específicos. La **discográfica** cuenta con todo el catálogo de álbumes editados por ella hasta el momento.

Por cada **álbum** se conoce su título, y género (para este dato, existe la condición excluyente que sólo se editen discos de estos tres géneros: GRUNGE, ELECTRONICA Y TRAP).

De cada **artista** se conoce su nombre, apellido, nombre artístico y el rol que desempeña dentro de la banda: CANTANTE, GUITARRISTA, BATERISTA, BAJISTA, TECLADISTA o NINGUNO.

Cada **álbum** tiene una colección de temas que lo integran. De cada uno se registra el nombre, su duración en segundos, un artista que es el único compositor del tema (el cual puede o no tocar un instrumento) y una colección con los artistas que participan en la ejecución del tema (si el compositor además canta o cumple otro rol en el tema aparece junto a los demás artistas).

Se pide desarrollar:

La explotación de los siguientes métodos en NS+:

Constructor de la clase Álbum.

Método eliminarTema(...) que recibe el nombre de un tema (String) y un nombre artístico del compositor (String) y debe eliminarlo de todos los álbumes donde se encuentre. Tener en cuenta que puede haber más de un tema con el mismo nombre y sólo debe eliminarse el que corresponda al compositor indicado por parámetro. El método debe devolver un objeto que contendrá el tema eliminado y todos los álbumes en los que se encontró (no sólo los nombres).

El método mostrarAlbumesPorGenero(...) que recibe un Genero y debe mostrar por pantalla todos los álbumes editados de un determinado género musical con el formato que se muestra a continuación:

Album: “Nevermind”

Duración Total: 2980 segundos

Temas:

Nombre del Tema: “Come as you are”

Duración: 180 segundos

Compositor: Kurt Cobain, GUITARRISTA

Artistas:

Kurt Cobain, GUITARRISTA

Dave Grohl, BATERISTA

2do Final Diciembre 2021

Enunciado

“Tester de Aplicaciones”

TeamQA es una empresa especializada en el testeo de aplicaciones. Esta empresa necesita desarrollar un programa Tester que permita probar automáticamente cualquier programa en base a una serie de pruebas.

El **Tester** guarda un registro histórico con los resultados de todas las aplicaciones probadas hasta el momento. En cada resultado conoce la Aplicación probada y el valor del resultado de la prueba:

PRUEBA_SUPERADA, cuando toda la serie de pruebas se cumplió exitosamente.

PRUEBA_SUPERADA_CON_FALLOS, cuando hubo pruebas de la serie que no se superaron pero donde éstas no fueron marcadas como stoppers (tan importantes que no tiene sentido seguir con la prueba).

PRUEBA_NO_SUPERADA. Cuando falló una prueba considerada como stopper y entonces la serie de pruebas no se completó.

De la Aplicación a testear se conoce su nombre y su versión.

Cuando se ejecuta un test, el Tester recibe la aplicación a probar y la serie de pruebas a realizar (una colección).

Cada Prueba de la serie tiene una marca que indica si es stopper y una serie de pasos (PasoDePrueba) a cumplir. Si la prueba es stopper y algún paso de la prueba no termina en OK se la considera como una prueba no superada y el test completo se interrumpe.

Una instancia de PasoDePrueba únicamente tiene un código y un método llamado probar(...) el cual recibe como parámetro la aplicación que se está probando y nada más (todo el resto de su información es privada y no nos interesa). Este método devuelve uno de estos resultados:

OK: cuando se obtiene el resultado esperado;

FALLO: cuando no se obtiene el resultado esperado;

TIMEOUT: cuando sin obtener resultados se cumple el tiempo determinado para su ejecución.

Sólo se debe seguir con la ejecución de una prueba mientras el resultado de la ejecución de cada uno de sus pasos resulte OK.

Nota:

El método probar(...) no debe ser implementado en NS+, solamente usado donde haga falta.

Se pide:

Confeccionar el diagrama UML que describe el escenario del enunciado incluyendo los atributos de cada clase y los métodos a desarrollar y aquellos que creas conveniente.

Los constructores de las clases Tester y Prueba.

Desarrollar el método ejecutarTesting(...) de la clase Tester que, recibiendo como parámetro la aplicación y una serie de pruebas, ejecute las pruebas sobre la aplicación hasta cumplirlas todas o detenerse por la ejecución no exitosa de una prueba stopper.

Al finalizar la prueba debe generarse y agregarse el Resultado correspondiente en el histórico.

Desarrollar todos los métodos necesarios para complementar el método anterior (salvo el método probar() de la PasoDePrueba).

final julio 2022

- Correcta implementación de constructores.
- Modularización reutilizable y mantenible con uso de métodos con correcta parametrización y correcto encapsulamiento, publicando *setters* y *getters* sólo cuando corresponda.
- Manejo de clases, enumerados y colecciones (con todo lo que esto implica).

Enunciado

YaEmpanadas es una aplicación que complementa el trabajo de una cadena existente en esta ciudad.

La aplicación registra y guarda cada uno de los pedidos que recibe. Cada pedido tiene el número de teléfono de quien hace el pedido y una dirección física donde se enviarán las empanadas.

También, cada pedido tiene uno o más ítems. En cada ítem se guarda la cantidad de empanadas que se piden de cada sabor. Los sabores disponibles son fijos: CARNE, POLLO, JAMÓN Y QUESO, VERDURA, HUMITA.

Se pide:

- Diseñar el diagrama UML completo que resuelva el enunciado propuesto.
- Desarrollar en NS+ los siguientes puntos (incluyendo los métodos derivados):
 - El constructor de **YaEmpanadas**.
 - Estos métodos públicos:
 - ~~crearORecuperarPedido: Recibe un teléfono y una dirección. A partir de estos datos, busca un pedido existente para ese teléfono. Si no lo encuentra, lo registra. Devuelve el número del pedido, el cual será necesario para la carga de las empanadas y para mostrar el detalle del pedido. El número de pedido se debe incrementar automáticamente con cada pedido creado.~~
 - cargarEmpanadas: Recibe un número de pedido (el pedido debe existir), un sabor y una cantidad (debe ser mayor que cero).
 - Carga la cantidad de unidades y el sabor en un ítem del pedido según estas reglas:
 - Si el ítem existe (uno con el mismo sabor), **reemplaza** el valor actual por el nuevo. Solo acepta cantidades mayores que cero.
 - Si el ítem no existe, lo agrega a los existentes. También debe controlar la cantidad de unidades (como mínimo una empanada).
 - Si hay un error debe emitir un mensaje de error acorde al error producido. Si todo salió bien debe mostrar un aviso de que el pedido fue actualizado.
 - listarPedidoCompleto: Recibe un número de pedido y lista toda la información referida a este pedido. Si el pedido no existiera debe mostrarse un mensaje de error.
 - Desarrollar también los métodos privados y/o públicos de las clases existentes que creas necesarios para resolver lo pedido.