



UNIVERSIDAD AUTÓNOMA DE ZACATECAS

---

## Practica 11: Listas simplemente ligadas

---

*Estudiante:*

José Francisco Hurtado Muro

*Profesor:*

Dr. Aldonso Becerra Sánchez

May 4, 2025

# Tabla de Contenidos

<b>1</b>	<b>Introducción</b>	<b>2</b>
<b>2</b>	<b>Actividades que debe realizar el alumno:</b>	<b>3</b>
2.1	Actividad inicial: . . . . .	3
2.2	Actividad 1: . . . . .	3
2.3	Actividad 2: . . . . .	3
2.3.1	TDA - PilaDinamica . . . . .	3
2.4	Actividad 3: . . . . .	5
2.4.1	TDA - ColaDinamica . . . . .	5
<b>3</b>	<b>Actividad 4:</b>	<b>7</b>
3.1	imprimirDesc() . . . . .	7
3.2	encontrarLista(Object valor) . . . . .	8
3.3	aListaEstatica() . . . . .	9
3.4	aMatriz2D(int filas, int columnas) . . . . .	10
3.5	agregarLista(ListaDatos listaDatos2) . . . . .	11
3.6	Clonar() . . . . .	12
3.7	boolean agregarMatriz2D(Arreglo2D tabla, TipoTabla enumTipoTabla) . .	13
3.8	void vaciar() . . . . .	14
3.9	void rellenar(Object valor, int cantidad) . . . . .	15
3.10	int contar(Object valor) . . . . .	15
3.11	void invertir(). . . . .	16
<b>4</b>	<b>Código Agregado - UML</b>	<b>18</b>
<b>5</b>	<b>Pre-evaluación del Alumno</b>	<b>19</b>
<b>6</b>	<b>Conclusión</b>	<b>19</b>
<b>7</b>	<b>Referencias:</b>	<b>19</b>

# 1 Introducción

"La memoria dinámica es un elemento importante en el manejo de información abundante donde no se sabe de antemano cuantos datos son los requeridos, por tanto solventa las limitaciones de la memoria estática. Las listas enlazadas permiten la manipulación de la memoria dinámica a través de la liga de nodos sucesivos."

## 2 Actividades que debe realizar el alumno:

### 2.1 Actividad inicial:

Generar el reporte en formato **IDC**.

### 2.2 Actividad 1:

Primero genere la **Introducción**.

### 2.3 Actividad 2:

Implemente la funcionalidad de la clase Pila utilizando listas dinámicas en lugar de arreglos (llamada PilaDinamica). Invoque métodos existentes.

Haga el programa (actividad 2, la cual es el **Desarrollo** del programa, junto con la captura de pantalla del programa funcionando).

#### 2.3.1 TDA - PilaDinamica

Recordemos que en un previo uso de PilaFija tenia la limitante de que era limitada, ya que en vez de usar una lista Dinámica / lista Ligada usaba un Arreglo para guardar los datos, por lo cual ahora se creo esta variante de Pila que resuelve ese problema, donde ahora puede ser de un tamaño indefinido y que va poco a poco pidiendo espacio en memoria según vayamos agregando valores en ella, por esto mismo hubo una reestructuración en los métodos pero siguiendo con la esencia de una Pila, como podemos ver en la figura 2.

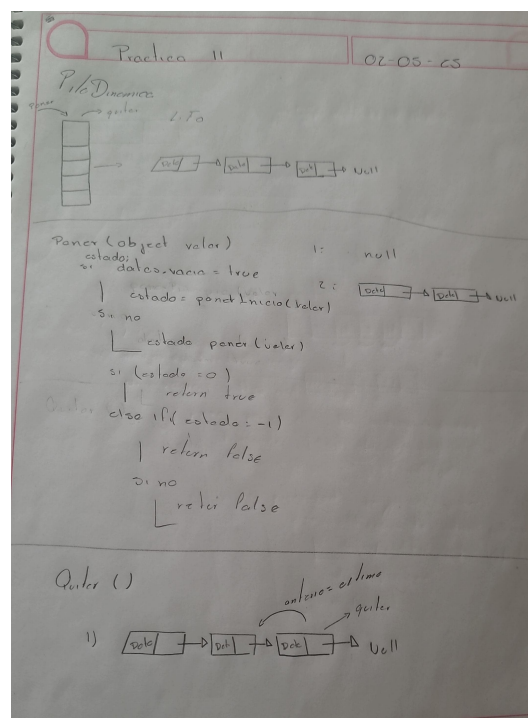


Figure 1: Análisis: métodos del TDA - PilaDinamica

### Explicación

Para este caso podemos ver como se cambio ciertas partes de los métodos, pero siguen teniendo la lógica de como funciona una pila, solo que en este caso se uso una ListaDin que varios de sus métodos ya funcionan muy similar a ciertas cosas que caracteriza a una pila, en especial en sus métodos mas importantes como lo son **Quitar, Poner y verTope**, como lo pedemos ver en la figura 1 y figura 2.

```

public class PilaDinamica {
    protected ListaDin datos;

    public PilaDinamica() { // inicia la pila dinamica que es de tama
        datos = new ListaDin();
    }

    public boolean vacia() { // utiliza el metodo de vacio de la Lis
        return datos.vacio();
    }

    //agregar un valor a la pila por medio del metodo ponerInicio o
    public boolean poner(Object valor) {
        int estado;
        if( vacia() == true){
            estado = datos.ponerInicio(valor); //si esta vacia se por
        }else{
            estado = datos.poner(valor); // si no esta vacia se pone
        }

        if (estado == 0){
            return true; // si si lo agregp
        }else if(estado == -1){
            return false; // si no lo agrego
        }else{
            return false; // por si no pasa nada de lo anterior
        }
    }

    // se quita el ultimo elemento que entro a la pila por medio del
    public Object quitar() {
        return datos.quitar(); // quita el ultimo nodo y recorre al
    }

    public void imprimir() {
        datos.imprimir();
    }

    //muestra el ultimo elemento ingresado a la pila
    public Object verTope() {
        return datos.verFinal();
    }
}

```

```

PS C:\Users\Josef\OneDrive\Documentos\mis_doc
:\Program Files\Java\jdk-21\bin\java.exe' '-X
77d1208e4045857c\redhat.java\jdt_ws\Estructur
¿La pila está vacía? true
Agregando elementos a la pila...
¿La pila está vacía? false
Elemento en el tope de la pila: 30
Elementos en la pila:
10 -> 20 -> 30 -> null
Quitando elementos de la pila...
Elemento quitado: 30
Elemento quitado: 20
Elemento en el tope de la pila: 10
Elementos restantes en la pila:
10 -> null
Quitando el último elemento...
Elemento quitado: 10
¿La pila está vacía? true
Intentando quitar de una pila vacía: null
PS C:\Users\Josef\OneDrive\Documentos\mis_doc

```

Figure 2: Funcionamiento: métodos TDA - PilaDinamica

## 2.4 Actividad 3:

Implemente la funcionalidad de la clase Cola simple utilizando listas dinámicas en lugar de arreglos (llamada ColaDinamica). Invoque métodos existentes.

Haga el programa (actividad 3, la cual es el **Desarrollo** del programa, junto con la captura de pantalla del programa funcionando).

### 2.4.1 TDA - ColaDinamica

Tengamos en cuenta que se tenía una versión previa, ColaFija, la cual tenía una limitante, la cual era un tamaño fijo, con esta versión se repara esta limitante del tamaño con el uso de listas dinámicas, donde se usa los métodos de esta estructura de datos, solo que en este caso para respetar los principios característicos de una Cola, se cambio la forma de quitar elementos de la cola, ya que en vez de quitar el ultimo, se quita el primero ya que funciona con el principio FiFo, como podemos ver la figura 3 y 4.

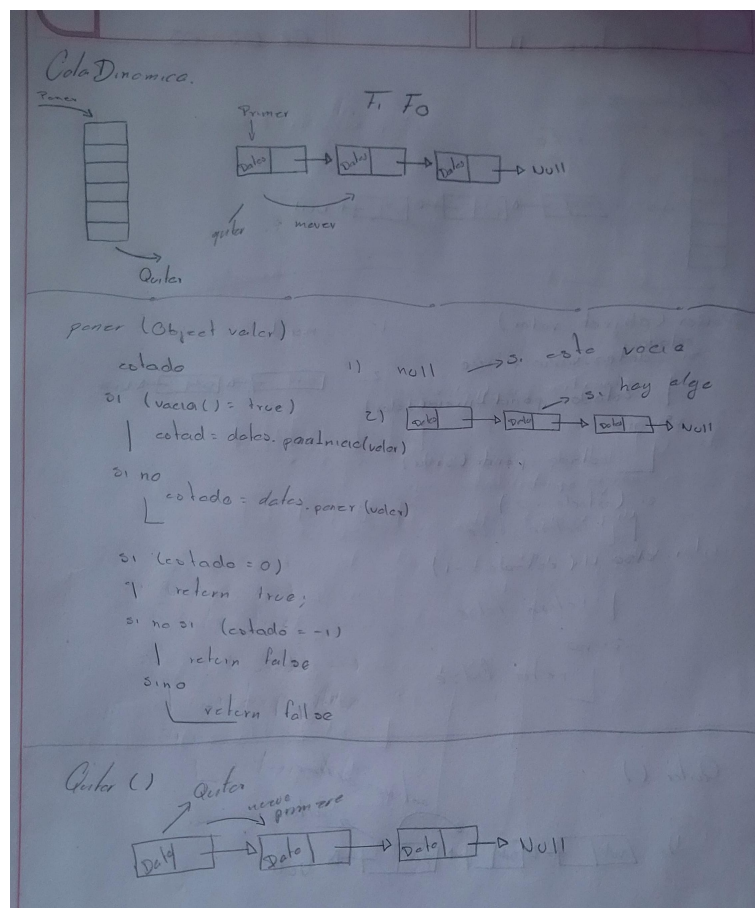


Figure 3: Análisis: métodos del TDA - PilaDinamica

### Explicación

En este caso, en lugar de usar un arreglo estático para almacenar los datos, se optó por una lista dinámica. La ventaja es que la lista va solicitando memoria según se necesite, evitando así reservar espacio de antemano. Esto mantiene la eficiencia y sigue cumpliendo con las reglas básicas de una Cola, especialmente en las operaciones clave como **poner** y **quitar**, como podemos ver en la figura 4.

```

public class ColaDinamica {
    protected ListaDin datos;

    public ColaDinamica(){
        datos = new ListaDin(); //lugar donde se va a guardar los da
    }

    // revisar si esta vacia
    public boolean vacia() {
        return datos.vacio();
    }

    // agregar datos a la lista dinamica
    public boolean poner(Object valor) {
        int estado;
        if( vacia()== true){
            estado = datos.ponerInicio(valor); //si esta vacia se pon
        }else{
            estado = datos.poner(valor); // si no esta vacia se pone
        }

        if (estado == 0){
            return true; // si si lo agregp
        }else if(estado== -1){
            return false; // si no lo agrego
        }else{
            return false; // por si no pasa nada de lo anterior
        }
    }

    public Object quitar() {
        return datos.quitarInicio();
    }

    public void imprimir() {
        datos.imprimir();
    }

    public Object verTope() {
        return datos.verInicio();
    }
}

```

```

e29d877d1208e4045857c\redhat.java\jdt_ws\
¿La cola está vacía? true
Agregando elementos a la cola...
¿La cola está vacía? false
Elemento en el frente de la cola: 10
Elementos en la cola:
10 -> 20 -> 30 -> null
Quitando elementos de la cola...
Elemento quitado: 10
Elemento quitado: 20
Elemento en el frente de la cola: 30
Elementos restantes en la cola:
30 -> null
Quitando el último elemento...
Elemento quitado: 30
¿La cola está vacía? true
Intentando quitar de una cola vacía: null
PS C:\Users\Josef\OneDrive\Documentos\Mis

```

Figure 4: Funcionamiento: métodos TDA - PilaDinamica

### 3 Actividad 4:

Implemente el código de listas enlazadas (la lista no debe tener atributos que lleven el control numérico de cuántos elementos existen actualmente en la lista):

#### 3.1 imprimirDesc()

Imprimir en orden inverso los elementos de la clase ListaDin (implementa ListaDatos): void mostrarOI(). Use pilas para resolverlo.

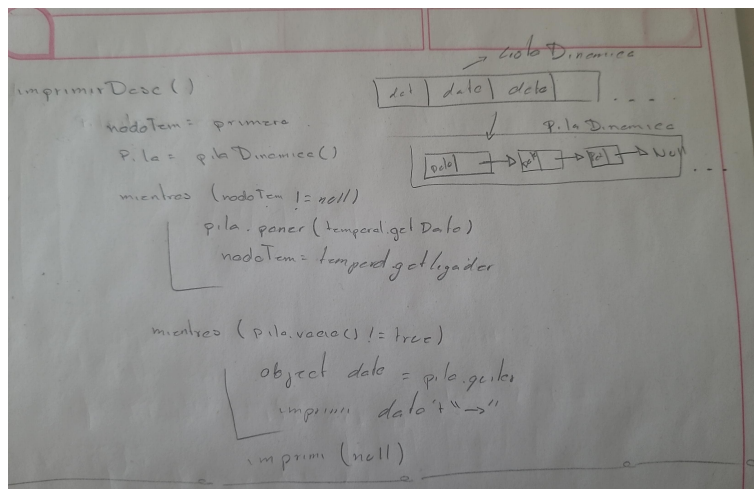


Figure 5: Análisis: void imprimirDesc()

#### Explicación

En este caso lo que se hizo fue pasar todo lo de una lista dinamica a una pila dinamica, así evitamos el uso de un contador de cuantos datos existen en la lista dinamica, una vez con todos los datos en la pila dinamica, se vacía esta para poder imprimir los datos de manera inversa esto gracias al principio **LiFo**, como podemos ver en la figura 6.

```

run | Debug
public static void main(String[] args) {
    ListaDin lista = new ListaDin();
    lista.poner(valor:"a");
    lista.poner(valor:"x");
    lista.poner(valor:"m");
    lista.poner(valor:"r");
    lista.poner(valor:"y");
    lista.poner(valor:"r");
    lista.poner(valor:"g");
    lista.poner(valor:"d");

    Salida.salidaPorDefecto(cadena:"Inverso\n");
    lista.imprimirDesc();
    Salida.salidaPorDefecto(cadena:"\nnormal\n");
    lista.imprimir();
}

Inverso
d -> g -> r -> y -> r -> m -> x -> a -> null

normal
a -> x -> m -> r -> y -> r -> g -> d -> null
PS C:\Users\Josef\OneDrive\Documentos\Mis_docs\4
    
```

Figure 6: Funcionamiento: void imprimirDesc()



### 3.2 encontrarLista(Object valor)

Realizar el método que permita encontrar todas las ocurrencias de un elemento en una lista enlazada. El método debe regresar el contenido de los valores encontrados como lista: ListaDin encontrarLista(Object valor). Lo interesante de este método es que se debe analizar cómo se puede reutilizar hacia métodos existentes que usan su funcionalidad.

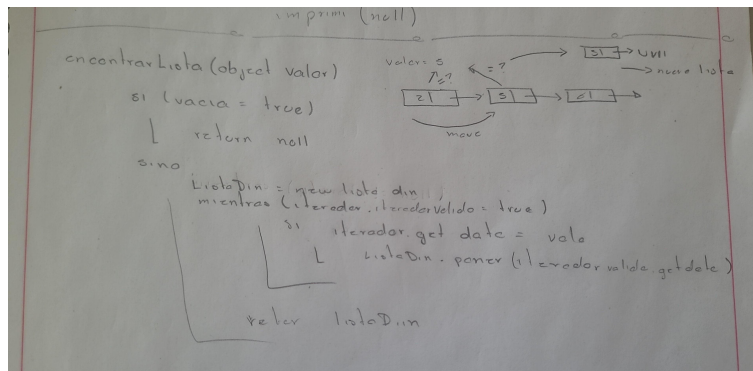


Figure 7: Análisis: ListaDin encontrarLista(Object valor)

#### Explicación

Para este caso lo que se hizo fue revisar que la lista original no este vacia, de serlo asi, regresa null, pero en caso de no estar vacia lo que se hace es instacia una nueva lista que sera el retorno, ademas de que se iniciaría una iterador, donde si este se mantiene como valido, seguirá en pie el ciclo, y donde encuentre un dato del iterador que sea igual al valor que nosotros estemos buscando, lo copiara en la nueva lista dinámica y funcionaria como podemos ver en la figura 8.

```

ListaDin lista = new ListaDin();
lista.poner(valor:"a");
lista.poner(valor:"x");
lista.poner(valor:"m");
lista.poner(valor:"r");
lista.poner(valor:"y");
lista.poner(valor:"g");
lista.poner(valor:"g");
lista.poner(valor:"g");

/*
Salida.salidaPorDefecto("inverso\n");
lista.imprimirDesc();
Salida.salidaPorDefecto("\nnormal\n");
lista.imprimir();
*/

ListaDin listaNueva = lista.encontrarLista(valor:"g");
listaNueva.imprimir();

```

Figure 8: Funcionamiento: ListaDin encontrarLista(Object valor)

### 3.3 aListaEstatica()

Hacer un método que guarde todos los elementos de una lista dinámica en una Arreglo, el cual debe regresar como valor de retorno: Arreglo aListaEstatica().

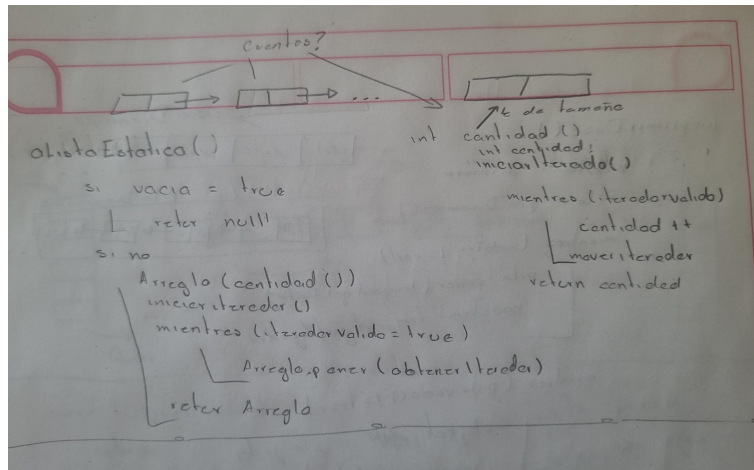


Figure 9: Análisis: Arreglo aListaEstatica()

#### Explicación

Lo que se hace en este caso, es revisar que la lista no este vacía, en caso de serlo no hace nada, lo siguiente es que si no esta vacía, va a instanciar una lista estática (**un Arreglo**) y este tendrá un tamaño que se obtendrá con ayuda de un método que cuenta los nodos de la lista dinámica, para luego iterar sobre la lista dinámica y ir pasando los datos a la lista Estática, este proceso lo podemos ver en la figura 9 y 10.

```

1  static void main(String[] args) {
2      lista.poner(valor:"x");
3      lista.poner(valor:"m");
4      lista.poner(valor:"r");
5      lista.poner(valor:"y");
6      lista.poner(valor:"g");
7      lista.poner(valor:"g");
8      lista.poner(valor:"g");
9
10     /*
11     Salida.salidaPorDefecto("inverso\n");
12     lista.imprimirDesc();
13     Salida.salidaPorDefecto("\nnormal\n");
14     lista.imprimir();
15     */
16
17     /*
18     ListaDin listaNueva = lista.encontrarLista("g");
19     listaNueva.imprimir();
20     */
21
22     Arreglo arreglo = lista.aListaEstatica();
23     arreglo.imprimir();
24 }

```

Figure 10: Funcionamiento: Arreglo aListaEstatica()

### 3.4 aMatriz2D(int filas, int columnas)

Hacer un método que guarde los elementos de una lista en una matriz 2d. A este método se le pasarán el número de renglones y columnas de la matriz resultante. En caso que no se ajusten de renglones o columnas con la cantidad de elementos de la lista, se deben rellenar con null: Arreglo2D aMatriz2D(int filas, int columnas).

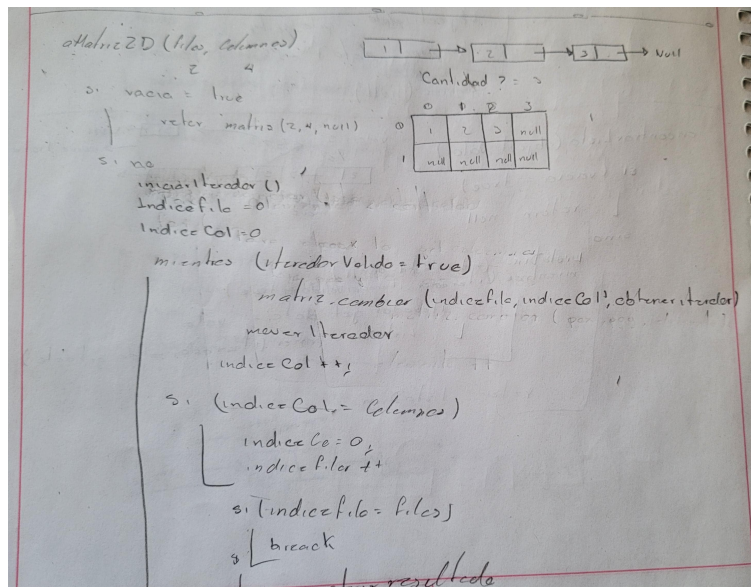


Figure 11: Análisis: Arreglo2D aMatriz2D(int filas, int columnas)

#### Explicación

Aquí lo que se realiza es que se instancia una matriz con un tamaño dicho, con todo su interior null, y si esta vacía la lista dinámica se regresa así con todo null, pero en caso de que no este vacía la lista dinámica lo que se hace es cambiar solo la cantidad de elementos según la lista dinámica, donde se recorre por columnas primero pero si llegamos al final la columna aumenta, pero las filas quedan en su mismo lugar, pero cuando se acaban las columnas se pasa de fila y volvemos a empezar pero cuando se acaban las filas se termina el proceso de cambio de valores y se regresa la matriz con los valores modificados como podemos ver en la figura 12.

```

lista2.poner(valor:1);
lista2.poner(valor:2);
lista2.poner(valor:3);

Arreglo2D arre2d = lista2.aMatriz2D(filas:2,columnas:4);
arre2d.imprimirXColumnas();
    
```

Figure 12: Funcionamiento: Arreglo2D aMatriz2D(int filas, int columnas)

### 3.5 agregarLista(ListaDatos listaDatos2)

Hacer un método que agregue al final de la lista los elementos pasados como argumentos en una Arreglo o una ListaDin (listadatos2). boolean agregarLista(ListaDatos listaDatos2).

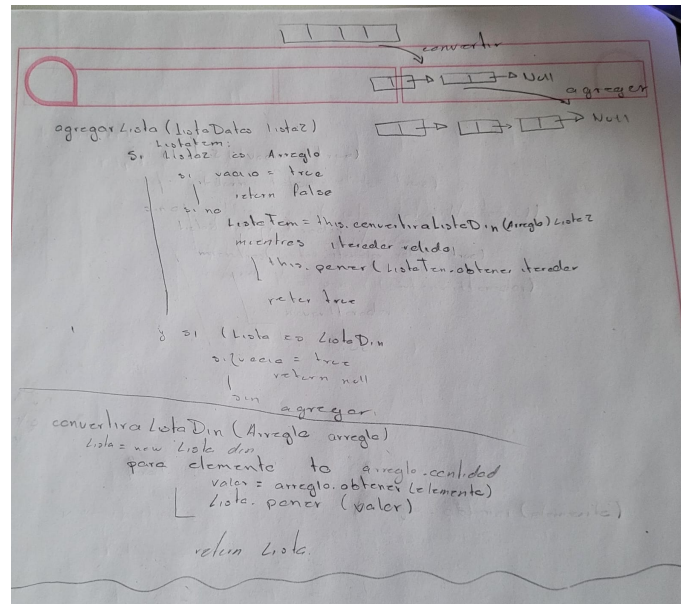


Figure 13: Análisis: boolean agregarLista(ListaDatos listaDatos2)

#### Explicación

En este caso lo que se hace es diferenciar los tipos de datos que se pueden enviar (**Arreglo** o **ListaDin**), con eso se podrán trabajar de manera diferente, si es arreglo se convertirá en ListaDin, de esta manera podemos hacer tratamientos por separado, si es arreglo se convierte en lista dinámica, pero si es listaDinamica se queda así, pero de esta manera podemos unificar la manera en la que se agregan los datos en la lista dinámica actual, como podemos ver en la figura 14.

```

// Crear lista din
ListaDin lista1 = new ListaDin();
lista1.poner(valor:"A");
lista1.poner(valor:"B");
lista1.poner(valor:"C");

// Crear otra lista dinamica
ListaDin lista2 = new ListaDin();
lista2.poner(valor:"D");
lista2.poner(valor:"E");
lista2.poner(valor:"F");

// Crear un arreglo
Arreglo arreglo = new Arreglo(tamaño:3);
arreglo.poner(valor:"G");
arreglo.poner(valor:"H");
arreglo.poner(valor:"I");

// Probar agregar lista con otra lista dinamica
System.out.println("Antes de agregar lista2 a lista1:");
lista1.imprimir();

boolean resultado1 = lista1.agregarLista(lista2);
System.out.println("Resultado de agregar lista2 a lista1: " + resultado1);
lista1.imprimir();

// Probar agregar lista con un arreglo
boolean resultado2 = lista1.agregarLista(arreglo);
System.out.println("Resultado de agregar arreglo a lista1: " + resultado2);
lista1.imprimir();
    
```

Figure 14: Funcionamiento: boolean agregarLista(ListaDatos listaDatos2)

### 3.6 Clonar()

Hacer un método que devuelva una copia de la lista ligada. La primera lista debe ser independiente de la copia: clonar().

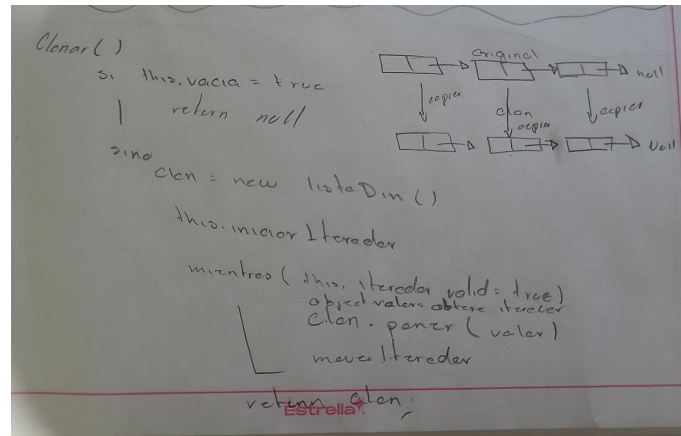


Figure 15: Análisis: Clonar()

#### Explicación

Para este caso lo que se hizo era revisar que la lista original no estuviera vacía, de serlo se regresa un null, pero si esta tiene contenido se instancia una lista dinámica donde se copia uno por uno los datos de la lista original, como podemos ver en la figura 16.

```

public static void main(String[] args) {
    ListaDin listaOriginal = new ListaDin();
    listaOriginal.poner(valor:"A");
    listaOriginal.poner(valor:"B");
    listaOriginal.poner(valor:"C");

    // Imprimir la lista original
    System.out.println("Lista original:");
    listaOriginal.imprimir();

    // Clonar la lista
    ListaDin listaClonada = (ListaDin) listaOriginal.clonar();

    // Imprimir la lista clonada
    System.out.println("Lista clonada:");
    listaClonada.imprimir();

    listaOriginal.poner(valor:"D");

    System.out.println("Lista original después de agregar un elemento:");
    listaOriginal.imprimir();

    System.out.println("Lista clonada después de modificar la original:");
    listaClonada.imprimir();
}

```

Figure 16: Funcionamiento: Clonar()



### 3.7 boolean agregarMatriz2D(Arreglo2D tabla, TipoTabla enumTipoTabla)

Hacer un método que agregue los elementos de una matriz 2d pasada como argumento al final de una lista. Los elementos irán agregando renglón por renglón o columna por columna, según se defina a través de una enumerado (COLUMNA; RENGLON): boolean agregarMatriz2D(Arreglo2D tabla, TipoTabla enumTipoTabla).

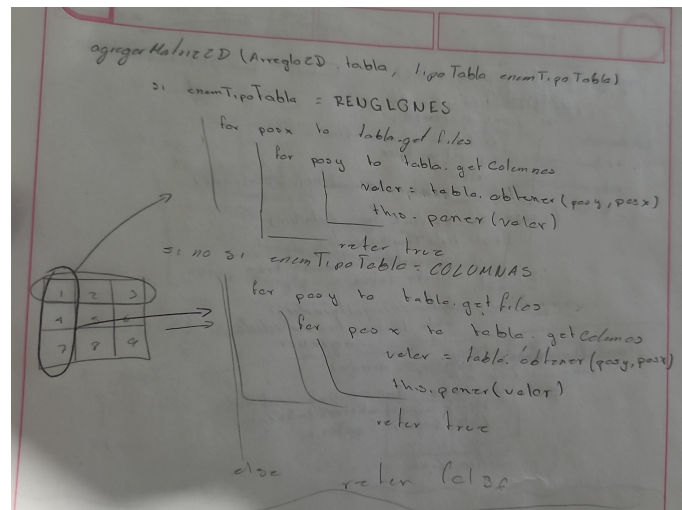


Figure 17: Análisis: boolean agregarMatriz2D(Arreglo2D tabla, TipoTabla enumTipoTabla)

#### Explicación

en este caso se dividió por dos casos RENGLÓN y COLUMNA, donde si es columna, se irán sacando los datos de la matriz por columna, y si es renglón se irán sacando por renglón, y de esta manera se irán colocando en la lista dinámica actual

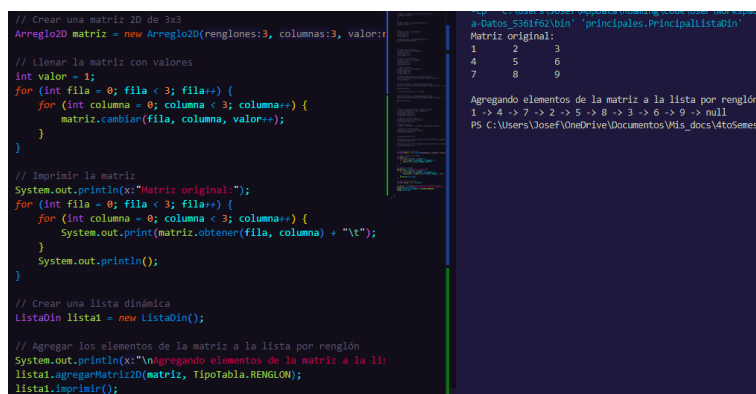


Figure 18: Funcionamiento: boolean agregarMatriz2D(Arreglo2D tabla, TipoTabla enumTipoTabla)

### 3.8 void vaciar()

Hacer el método que vacíe una lista ligada. void vaciar().

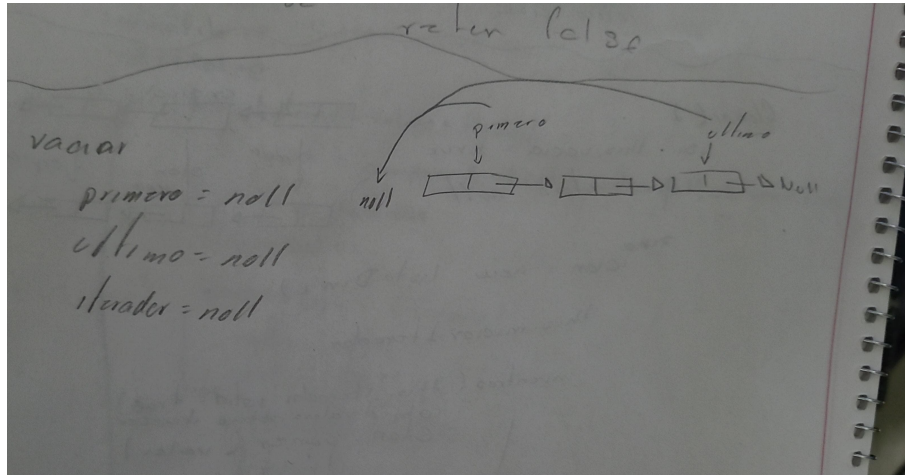


Figure 19: Análisis: void vaciar()

#### Explicación

Este caso es muy simple, ya que en este caso solo se tiene que apuntar las principales variables de ListaDin (**primero**, **ultimo** y **iterador**), a referencias null y quedaría limpia la listaDin, como podemos ver en la figura 20.

```

// Imprimir la lista antes de vaciarla
System.out.println(x:"Lista antes de vaciar:");
lista.imprimir();

// Vaciar la lista
lista.vaciar();

// Verificar si la lista está vacía
System.out.println("¿La lista está vacía? " + lista.vacio());

// Intentar imprimir la lista después de vaciarla
System.out.println(x:"Lista después de vaciar:");
lista.imprimir();

```

```

a:\jdc_ws\Estructura-Datos_5361f62\bin\principal
Lista antes de vaciar:
a -> x -> m -> r -> y -> g -> g -> g -> null
¿La lista está vacía? true
Lista después de vaciar:
null
PS C:\Users\Josef\OneDrive\Documents\Mis_docs\44

```

Figure 20: Funcionamiento: void vaciar()

### 3.9 void rellenar(Object valor, int cantidad)

Hacer un método que rellene una lista con valores iguales indicados por argumento. void rellenar(Object valor, int cantidad).

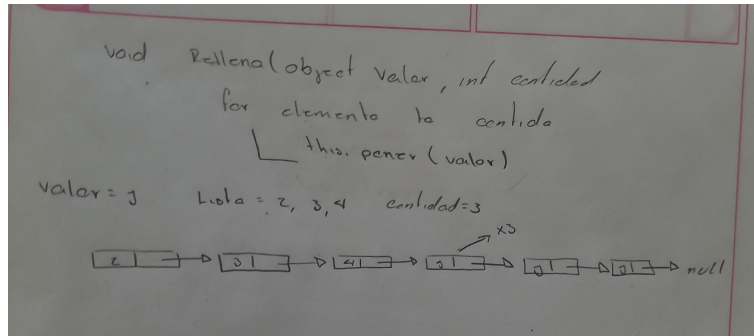


Figure 21: Análisis: void rellenar(Object valor, int cantidad)

#### Explicación

lo que se hizo aquí fue repetir el agregado del valor el numero de cantidad de veces dado, como podemos ver en la figura 22.

```
Salida.salidaPorDefecto(cadena:"original\n");
lista.imprimir();

lista.rellenar(valor:"j", cantidad:4);

Salida.salidaPorDefecto(cadena:"rellenada\n");
lista.imprimir();
```

```
original
a -> x -> m -> r -> null
rellenada
a -> x -> m -> r -> j -> j -> j -> j -> null
PS C:\Users\Josef\OneDrive\Documentos\Mis_docs
```

Figure 22: Funcionamiento: void rellenar(Object valor, int cantidad)

### 3.10 int contar(Object valor)

Hacer un método que cuente elementos en una lista con valores iguales indicados por el argumento. int contar(Object valor).

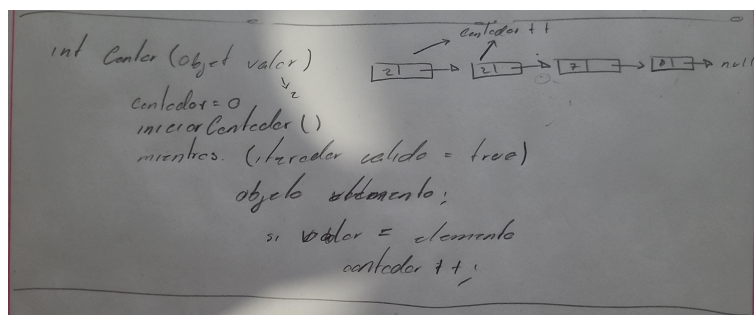


Figure 23: Análisis: int contar(Object valor)

#### Explicación



En este caso lo que se hace es iniciar un contador en 0, y este se incrementara si el valor buscado en la lista dinámica para después regresar lo, pero si la lista dinámica esta sola regresa un cero, tal y como podemos ver en la figura 23 y 24.

```
// Crear una lista dinámica
ListaDin lista = new ListaDin();
lista.poner(valor:"A");
lista.poner(valor:"B");
lista.poner(valor:"A");
lista.poner(valor:"C");
lista.poner(valor:"A");

// Imprimir la lista
System.out.println("Lista:");
lista.imprimir();

// Contar ocurrencias de "A"
int ocurrenciasA = lista.contar(valor:"A");
System.out.println("Ocurrencias de 'A': " + ocurrenciasA);

// Contar ocurrencias de "B"
int ocurrenciasB = lista.contar(valor:"B");
System.out.println("Ocurrencias de 'B': " + ocurrenciasB);

// Contar ocurrencias de un elemento que no está en la lista
int ocurrenciasD = lista.contar(valor:"D");
System.out.println("Ocurrencias de 'D': " + ocurrenciasD);
```

```
p C:\Users\Josef\AppData\Roam
atos_5361f62\bin' 'principales.
Lista:
A -> B -> A -> C -> A -> null
Ocurrencias de 'A': 3
Ocurrencias de 'B': 1
Ocurrencias de 'D': 0
PS C:\Users\Josef\OneDrive\Docu
```

Figure 24: Funcionamiento: int contar(Object valor)

### 3.11 void invertir().

Hacer un método que cuente elementos en una lista con valores iguales indicados por el argumento. int contar(Object valor).

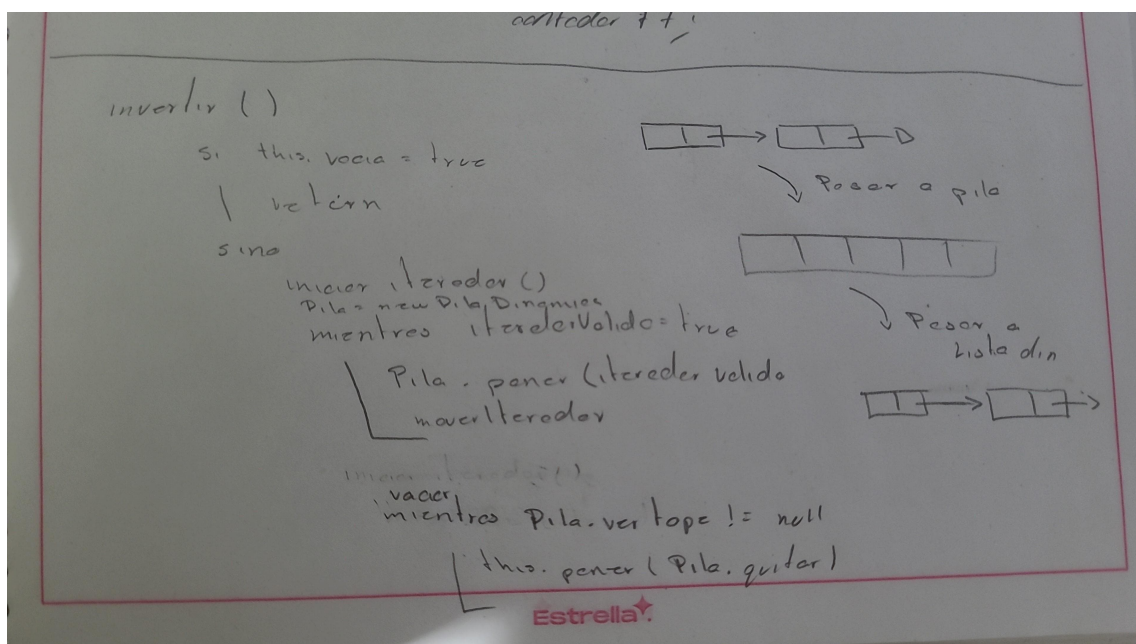
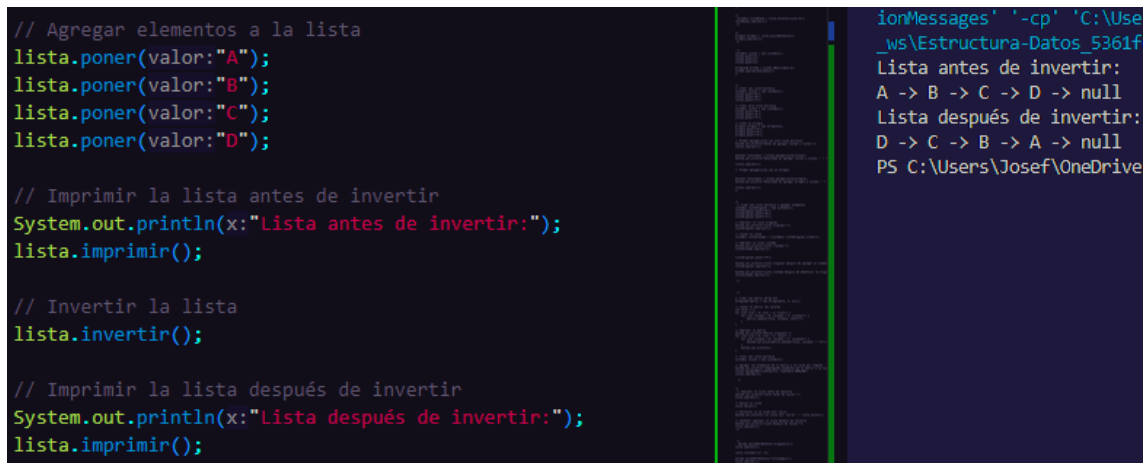


Figure 25: Análisis: void invertir().

### Explicación

Para este caso fue revisar que no estuviera sola la lista original, si lo estuviera ahí terminaría el método, pero si tuviera contenido se pasaría a una Pila Dinámica, una vez echo esto se vacía la lista original, y se pasan los datos de la pila a la lista Dinámica mientras el tope de la pila dinámica sea diferente a null, como podemos ver en la figura 25 y 26.



```
// Agregar elementos a la lista
lista.poner(valor:"A");
lista.poner(valor:"B");
lista.poner(valor:"C");
lista.poner(valor:"D");

// Imprimir la lista antes de invertir
System.out.println(x:"Lista antes de invertir:");
lista.imprimir();

// Invertir la lista
lista.invertir();

// Imprimir la lista después de invertir
System.out.println(x:"Lista después de invertir:");
lista.imprimir();
```

```
ionMessages' '-cp' 'C:\Use
_ws\Estructura-Datos_5361f
Lista antes de invertir:
A -> B -> C -> D -> null
Lista después de invertir:
D -> C -> B -> A -> null
PS C:\Users\Josef\OneDrive
```

Figure 26: Funcionamiento: void invertir().

## 4 Código Agregado - UML

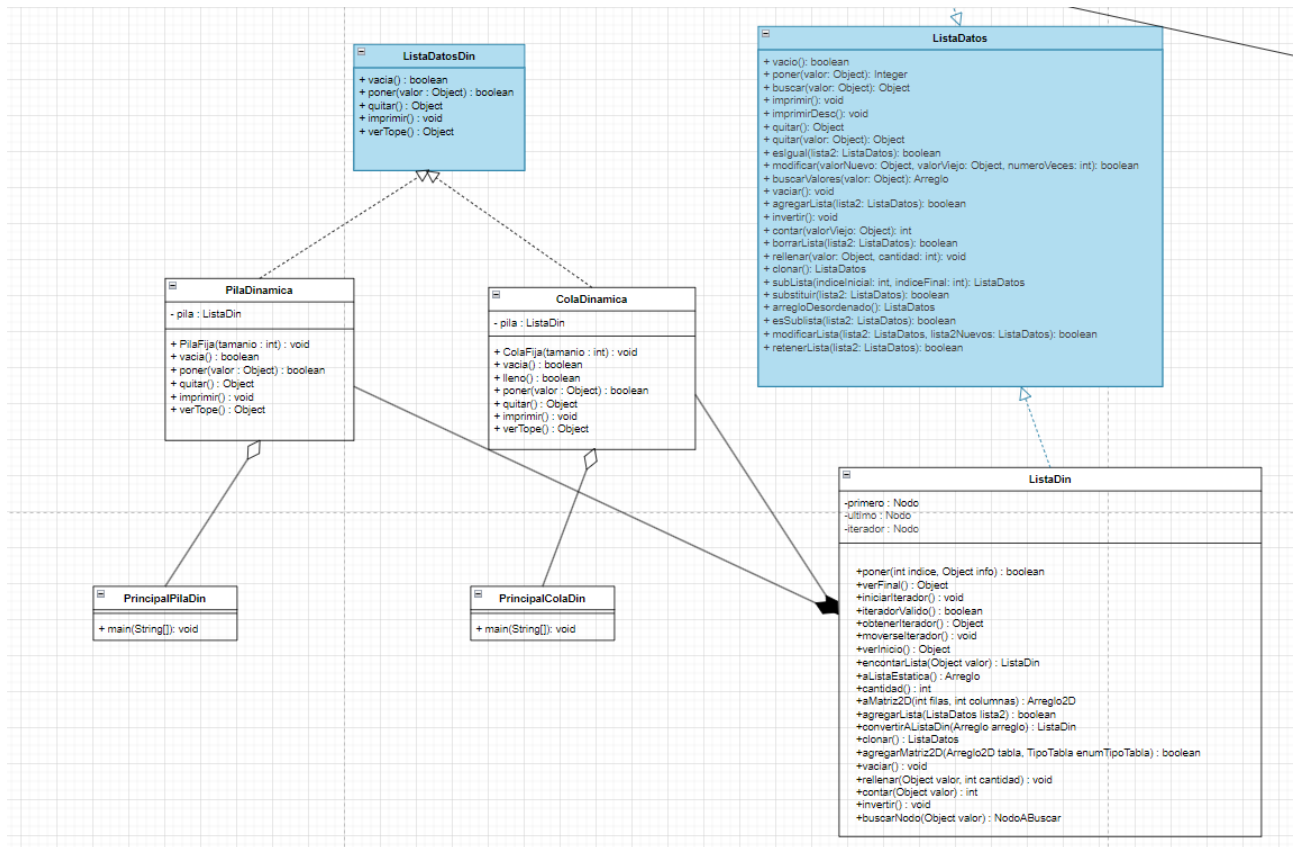


Figure 27: UML

## 5 Pre-evaluación del Alumno

Criterio	Evaluación
Cumple con la funcionalidad solicitada	<b>Sí</b>
Dispone de código auto-documentado	<b>Sí</b>
Dispone de código documentado a nivel de clase y método	<b>Sí</b>
Dispone de indentación correcta	<b>Sí</b>
Cumple la POO	<b>Sí</b>
Dispone de una forma fácil de utilizar el programa para el usuario	<b>Sí</b>
Dispone de un reporte con formato IDC	<b>Sí</b>
La información del reporte está libre de errores de ortografía	<b>Sí</b>
Se entregó en tiempo y forma la práctica	<b>No</b>
Incluye el código agregado en formato UML	<b>Sí</b>
Incluye las capturas de pantalla del programa funcionando	<b>Sí</b>
La práctica está totalmente realizada (especifique el porcentaje completado)	<b>87%</b>

Table 1: Evaluación de la práctica

## 6 Conclusión

mi estado mental esta arruinado ayuda

## 7 Referencias:

- Cairo, Osvaldo; Guardati, Silvia. *Estructura de Datos, Tercera Edición*. McGraw-Hill, México, Tercera Edición, 2006.
- Mark Allen Weiss. *Estructura de datos en Java*. Ed. Addison Wesley.
- Joyanes Aguilar, Luis. *Fundamentos de Programación. Algoritmos y Estructuras de Datos*. Tercera Edición, 2003. McGraw-Hill.