



UNIVERSIDAD AUTÓNOMA DE ZACATECAS

Practica

Estudiante:

José Francisco Hurtado Muro

Profesor:

Dr. Aldonso Becerra Sánchez

May 11, 2025

Tabla de Contenidos

1	Actividades que debe realizar el alumno:	3
1.1	Actividad inicial:	3
1.2	Actividad 1:	3
1.3	Actividad 2:	3
1.3.1	NodoClaveValor	3
1.4	TDA - ListaDinClave	4
1.4.1	boolean poner(Object clave, Object valor)	4
1.4.2	Object quitar(Object clave)	5
1.4.3	Object quitarContenido(Object valor)	6
1.4.4	Object buscar(Object clave)	7
1.4.5	Object buscarContenido(Object valor)	8
1.4.6	boolean cambiar(Object clave, Object valor)	8
1.4.7	boolean cambiarValor(Object valorViejo, Object valorNuevo)	9
1.4.8	void mostrar(), void mostrarClaves(), void mostrarValores()	10
1.4.9	Arreglo aListasEstaticas()	11
1.4.10	void vaciar()	12
1.4.11	Object obtener(Object clave)	12
1.4.12	boolean vacia()	13
1.4.13	int cantidad()	14
2	Código Agregado - UML	15
3	Pre-evaluación del Alumno	16
4	Conclusión	16
5	Referencias:	17

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."

1 Actividades que debe realizar el alumno:

1.1 Actividad inicial:

Generar el reporte en formato **IDC**.

1.2 Actividad 1:

Primero genere la **Introducción**.

1.3 Actividad 2:

Realizar la clase ListaDinClave, la cual consta de nodos de la siguiente estructura:

clave	valor	ligaDer
NodoClaveValor		

donde **clave** es un valor de referencia utilizado para realizar las operaciones sobre la lista (como si fuera una llave primaria o identificador único), y **valor** es el contenido real del nodo (el dato); **ligaDer** es el campo que indica el nodo que viene por delante en la lista.

1.3.1 NodoClaveValor

En este caso, **NodoClaveValor**, no puede ser un tipo mas específico de la clase **Nodo**, ya que este, en su atributo **ligaDer** es del tipo **Nodo**, y esto no nos serviría por que los nodos que queremos referenciar deben tener la referencia al nodo tipo **NodoClaveValor**, ya que este tiene lo un tercer elemento mas que el nodo normal y por medio de la herencia no estaremos trayendo ese campo que no necesitamos, talves pueda existir una posibilidad de que esta clase heredara, pero esto dependeria de como manejen el nodo, ya que se podria, por medio del polimorfismo y la sustitución de Liskov, pero en este caso para hacer mas simple el programa dejo que esta clase, fuera independiente como un objeto complejo, como podemos ver el diseño en la figura 1.

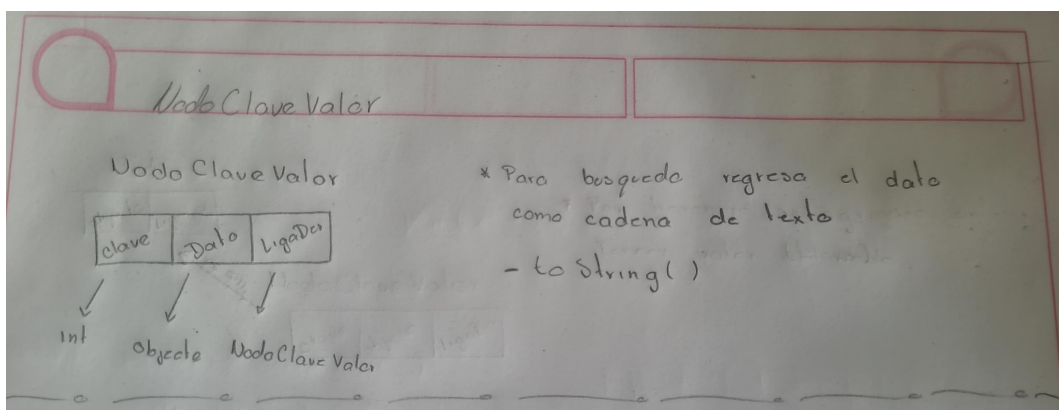


Figure 1: Estructura de NodoClaveValor

1.4 TDA - ListaDinClave

1.4.1 boolean poner(Object clave, Object valor)

Este método inserta los datos al final (se permiten duplicados en el campo de contenido, pero NO en el campo de clave). Si la clave ya se encuentra en la lista, el valor anterior será substituido por el nuevo valor con la misma clave.

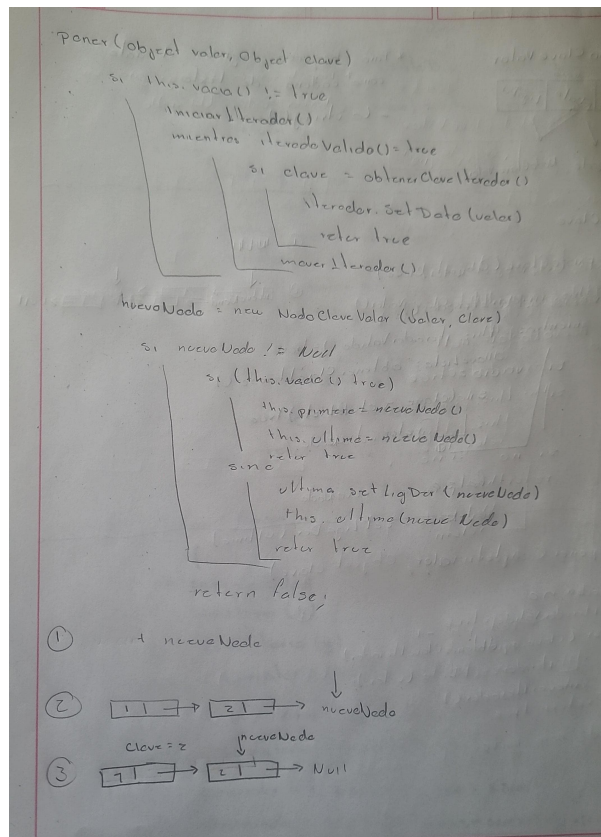


Figure 2: Análisis: poner(Object clave, Object valor)

Explicación

En este caso lo que se hizo, fue revisar iterar sobre la lista, buscando y comparando la clave, si la clave era la misma, en ese iterador/nodo, le cambiamos el Dato, pero se no haber similitud en las claves, se crea un nuevo nodo que pasa por dos procesos, donde si la lista es vacia se agrega al inicio, y el otro donde si no esta vacía la lista, el ultimo agrega la referencia de este nuevo nodo y el nuevo nodo pasa a ser el ultimo, tal y como podemos ver en la figura 2 y 3.

```
Salida.salidaPorDefecto(cadena:"Agregando elementos a la lista:\n");
lista.poner(valor:"Valor1", clave:"Clave1"); // Agregar un nodo con clave "Clave1"
lista.poner(valor:"Valor2", clave:"Clave2"); // Agregar un nodo con clave "Clave2"
lista.poner(valor:"Valor3", clave:"Clave3"); // Agregar un nodo con clave "Clave3"
// Imprimir la lista
Salida.salidaPorDefecto(cadena:"Contenido de la lista:\n");
lista.imprimir();
// Intentar agregar un nodo con una clave existente
Salida.salidaPorDefecto(cadena:"Reemplazando el valor de una clave existente:\n");
lista.poner(valor:"NuevoValor1", clave:"Clave1"); // Reemplazar el valor de "Clave1"
// Imprimir la lista
Salida.salidaPorDefecto(cadena:"Contenido de la lista:\n");
lista.imprimir();
```

```
-cp' 'C:\Users\Josef\AppData\Roaming\Code\User\ws
java\jdt_ws\Estructura-Datos_1c3df45c\bin' 'print
Agregando elementos a la lista:
Contenido de la lista:
Valor1 -> Valor2 -> Valor3 -> null
Reemplazando el valor de una clave existente:
Contenido de la lista:
NuevoValor1 -> Valor2 -> Valor3 -> null
PS C:\Users\Josef\OneDrive\Documents\Estructura-
```

Figure 3: funcionamiento: poner(Object clave, Object valor)

1.4.2 Object quitar(Object clave)

Se usa el campo de clave para borrar el elemento. Este método regresa el contenido real del campo. Regresa null si no se localiza.

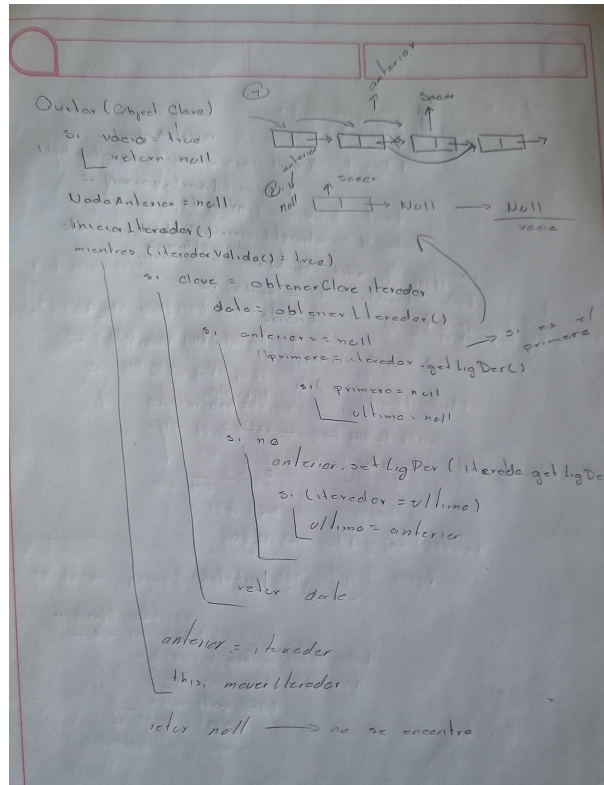


Figure 4: Análisis: quitar(Object clave)

Explicación

Para este caso lo que se hizo fue revisar primero que no estuviera vacía la lista, si esto fuera así, regresa un null, pero si esta tiene elementos, lo que se hace es crear un nodo, el cual contendrá el nodo anterior al removido, pero de primera instancia es nulo, por lo cual se procede a buscar la clave, si la clave coincide con alguna, ese nodo se remueve, dando al anterior la liga del posterior al nodo removido, tal y como vemos en la figura 4, pero he aquí donde tenemos dos casos, donde si el removido es el primer elemento se sabrá si el anterior se quedo en null, por lo cual se remueve el primero y a su liga derecha se convierte en el nuevo primero, pero si la liga derecha también es nula, significa que se a quedado vacía, así que se tiene que actualizar el ultimo a null, el otro caso es para si el nodo se encuentra entre medio o al final de la lista, donde sucede algo parecido, se une la liga derecha del anterior a la posterior del nodo a quitar, pero si fuera el ultimo nodo el que se quita se vuelve actualizar el ultimo nodo, dando así como que la liga derecha del anterior es null, y se vería como en la figura 5.

```
// Probar el método poner
Salida.salidaPorDefecto(cadena:"Agregando elementos a la lista:\n");
lista.poner(valor:"Valor1", clave:"Clave1"); // Agregar un nodo con clave "Clave1"
lista.poner(valor:"Valor2", clave:"Clave2"); // Agregar un nodo con clave "Clave2"
lista.poner(valor:"Valor3", clave:"Clave3"); // Agregar un nodo con clave "Clave3"
// Imprimir la lista
Salida.salidaPorDefecto(cadena:"Contenido de la lista:\n");
lista.imprimir();
// Intentar agregar un nodo con una clave existente
Salida.salidaPorDefecto(cadena:"Reemplazando el valor de una clave existente:\n");
lista.poner(valor:"NuevoValor1", clave:"Clave1"); // Reemplazar el valor de "Clave1"

// Imprimir la lista
Salida.salidaPorDefecto(cadena:"Contenido de la lista:\n");
lista.imprimir();

Salida.salidaPorDefecto(cadena:"eliminando clave3\n");
lista.quitar(clave:"Clave3");
lista.imprimir();
```

```
# Files\Java\jdk-21\bin\java.exe -XX:+ShowCodeStorage\cfad155481df61b9ad9ce3ab2f4bd7b\res
Agregando elementos a la lista:
Contenido de la lista:
Valor1 -> Valor2 -> Valor3 -> null
Reemplazando el valor de una clave existente:
Contenido de la lista:
NuevoValor1 -> Valor2 -> Valor3 -> null
eliminando clave3
NuevoValor1 -> Valor2 -> null
PS C:\Users\USER\OneDrive\Documents\estructura
```

Figure 5: funcionamiento: quitar(Object clave)

1.4.3 Object quitarContenido(Object valor)

Se usa el campo de valor para borrar el elemento. Este método regresa el contenido real del campo. Regresa null si no se localiza

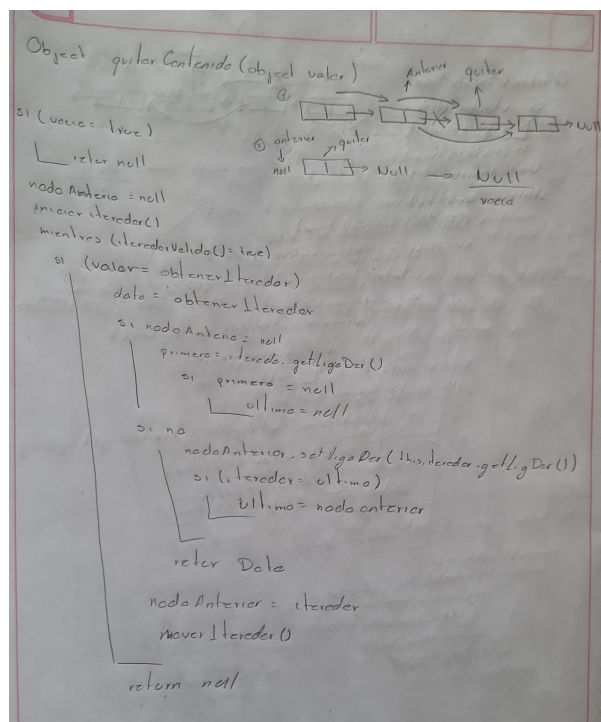
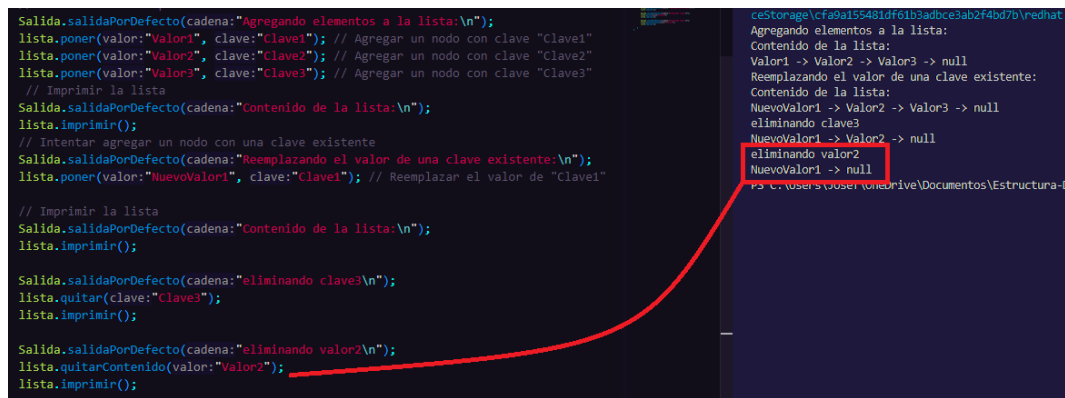


Figure 6: Análisis: quitarContenido(Object valor)

Explicación

En este caso este método es exactamente que el de **quitar(Object valor)**, ya que sigue los mismos principios, solo que aquí debe coincidir el dato dado con el dato que el dato de la lista, y si este coincide se hace exactamente lo mismo que el método **quitar(Object valor)**, de preferencia ir a leer la explicación de ese método, y funcionaria como se ve en la figura 7.



```

Salida.salidaPorDefecto(cadena:"Agregando elementos a la lista:\n");
lista.poner(Valor:"Valor1", clave:"Clave1"); // Agregar un nodo con clave "Clave1"
lista.poner(Valor:"Valor2", clave:"Clave2"); // Agregar un nodo con clave "Clave2"
lista.poner(Valor:"Valor3", clave:"Clave3"); // Agregar un nodo con clave "Clave3"
// Imprimir la lista
Salida.salidaPorDefecto(cadena:"Contenido de la lista:\n");
lista.imprimir();
// Intentar agregar un nodo con una clave existente
Salida.salidaPorDefecto(cadena:"Reemplazando el valor de una clave existente:\n");
lista.poner(Valor:"NuevoValor1", clave:"Clave1"); // Reemplazar el valor de "Clave1"

// Imprimir la lista
Salida.salidaPorDefecto(cadena:"Contenido de la lista:\n");
lista.imprimir();

Salida.salidaPorDefecto(cadena:"eliminando clave3\n");
lista.quitar(clave:"Clave3");
lista.imprimir();

Salida.salidaPorDefecto(cadena:"eliminando valor2\n");
lista.quitarContenido(Valor:"Valor2");
lista.imprimir();
    
```

```

ceStorage\cfa9a155481df61b3adbce3ab2f4bd7b\redhat...
Agregando elementos a la lista:
Contenido de la lista:
Valor1 -> Valor2 -> Valor3 -> null
Reemplazando el valor de una clave existente:
Contenido de la lista:
NuevoValor1 -> Valor2 -> Valor3 -> null
eliminando clave3
NuevoValor1 -> Valor2 -> null
eliminando valor2
NuevoValor1 -> null
    
```

Figure 7: funcionamiento: quitarContenido(Object valor)

1.4.4 Object buscar(Object clave)

Se utiliza para buscar un elemento ubicado por su clave. El método regresa el contenido en caso que se encuentre, null en caso contrario.

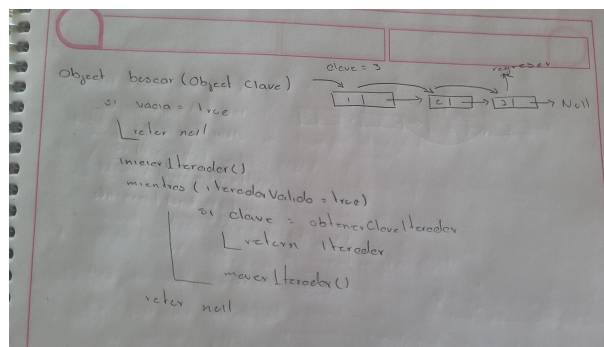



Figure 8: Análisis: buscar(Object clave)

Explicación

Aquí lo que se hizo fue ver si la lista estaba vacía, si lo estaba se regresa un null, pero de tener contenido se va recorriendo uno por uno y preguntándoles si la clave dada es igual a alguna ya existente en la lista, de serlo se regresa el dato de esa clave, como podemos ver en la imagen 9, y de no existir en la lista se regresa un null.



```

lista.imprimir();

Salida.salidaPorDefecto(cadena:"eliminando clave3\n");
lista.quitar(clave:"Clave3");
lista.imprimir();

Salida.salidaPorDefecto(cadena:"eliminando valor2\n");
lista.quitarContenido(Valor:"Valor2");
lista.imprimir();

Salida.salidaPorDefecto(cadena:"metiendo mas elementos\n");
lista.poner(Valor:"Valor2", clave:"Clave2"); // Agregar un nodo con clave "Clave2"
lista.poner(Valor:"Valor3", clave:"Clave3");
lista.imprimir();

Salida.salidaPorDefecto(cadena:"buscando clave3\n");
Salida.salidaPorDefecto(lista.buscar(clave:"Clave3")+"\n");
    
```

```

ceStorage\cfa9a155481df61b3adbce3ab2f4bd7b\redhat...
Agregando elementos a la lista:
Contenido de la lista:
Valor1 -> Valor2 -> Valor3 -> null
Reemplazando el valor de una clave existente:
Contenido de la lista:
NuevoValor1 -> Valor2 -> Valor3 -> null
eliminando clave3
NuevoValor1 -> Valor2 -> null
eliminando valor2
NuevoValor1 -> null
metiendo mas elementos
NuevoValor1 -> Valor2 -> Valor3 -> null
buscando clave3
Valor3
    
```

Figure 9: funcionamiento: buscar(Object clave)

1.4.5 Object buscarContenido(Object valor)

Se utiliza para buscar un elemento ubicado por su contenido. El método regresa el contenido en caso que se encuentre, null en caso contrario.

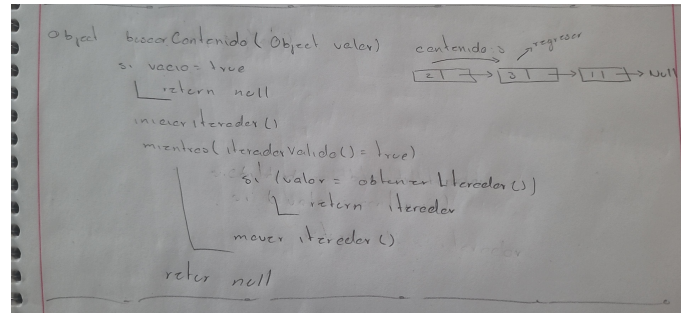


Figure 10: Análisis: buscarContenido(Object valor)

Explicación

Para este caso lo que se realizó, fue revisar que no estuviera vacía, si esta vacía esta regresa null, en caso contrario lo que se hizo fue iterar sobre la lista comparando los contenidos existentes y si este era igual, se regresaba, en caso de no encontrarlo se regresa null, como podemos ver en la figura 11.

```
Salida.salidaPorDefecto(cadena:"eliminando valor2\n");
lista.quitarContenido(valor:"Valor2");
lista.imprimir();

Salida.salidaPorDefecto(cadena:"metiendo mas elementos\n");
lista.poner(valor:"Valor2", clave:"Clave2"); // Agregar un nodo con clave "Clave2"
lista.poner(valor:"Valor3", clave:"Clave3");
lista.imprimir();

Salida.salidaPorDefecto(cadena:"buscando clave3\n");
Salida.salidaPorDefecto(lista.buscar(clave:"Clave3")+"\n");

Salida.salidaPorDefecto(cadena:"buscando valor3\n");
Salida.salidaPorDefecto(lista.buscarContenido(valor:"Valor3")+"\n");
```

Figure 11: funcionamiento: buscarContenido(Object valor)

1.4.6 boolean cambiar(Object clave, Object valor)

Substituye un elemento de la lista localizado mediante la clave por el nuevo valor. Regresa un booleano si pudo hacerlo

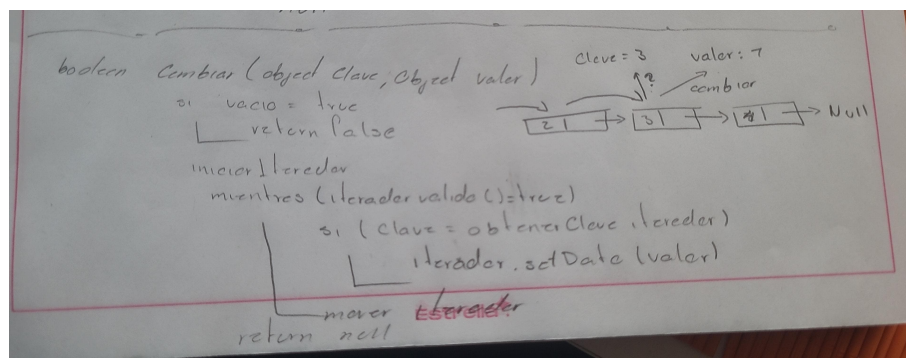


Figure 12: Análisis: cambiar(Object clave, Object valor)

Explicación

Aquí lo que se hizo fue ver que no estuviera vacía la lista, si estuviera vacía, se regresa un false, caso contrario, se iteraría sobre la lista y se estaría comparando las clave con la clave dada y si esta coincide, a ese nodo se le cambiara su contenido por el valor dado, de no encontrarse la clave se regresa un false, como podemos ver en la figura 13.

```

Salida.salidaPorDefecto(cadena:"Contenido de la lista:\n");
lista.imprimir();

Salida.salidaPorDefecto(cadena:"cambiando NuevoValor1 por valor1\n");
lista.cambiar(clave:"Clave1", valor:"Valor 1");
lista.imprimir();
    
```

buscando valor3
Valor2
Contenido de la lista:
NuevoValor1 -> Valor2 -> Valor3 -> null
cambiando NuevoValor1 por valor1
Valor 1 -> Valor2 -> Valor3 -> null

Figure 13: funcionamiento: cambiar(Object clave, Object valor)

1.4.7 boolean cambiarValor(Object valorViejo, Object valorNuevo)

Substituye un elemento de la lista localizado mediante el valor por el nuevo contenido. Regresa un booleano si pudo hacerlo.

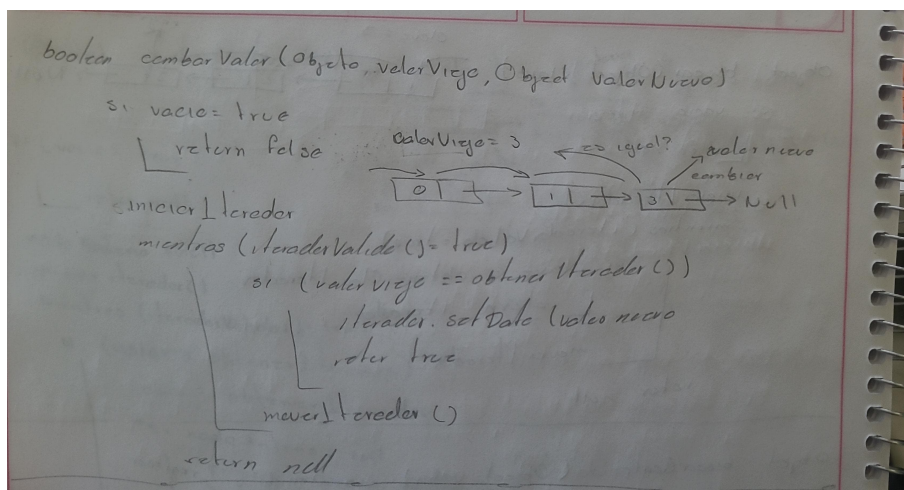


Figure 14: Análisis: cambiarValor(Object valorViejo, Object valorNuevo)

Explicación

Aquí lo que se hizo fue ver que no estuviera vacía la lista, si estuviera vacía, se regresa un false, caso contrario, se iteraría sobre la lista y se estaría comparando el valores con el valorViejo dado y si esta coincide, a ese nodo se le cambiara su contenido por el valor dado, de no encontrarse el valorViejo se regresa un false, como podemos ver en la figura 15.

```

Salida.salidaPorDefecto(cadena:"Contenido de la lista:\n");
lista.imprimir();

Salida.salidaPorDefecto(cadena:"cambiando NuevoValor1 por valor1, por conyenido \n");
lista.cambiarValor(valorViejo:"NuevoValor1", valorNuevo:"Valor 1");
lista.imprimir();
    
```

Valor3
Contenido de la lista:
NuevoValor1 -> Valor2 -> Valor3 -> null
cambiando NuevoValor1 por valor1, por conyenido \n
Valor 1 -> Valor2 -> Valor3 -> null
D5: C:\Users\jorgef\OneDrive\Documents\Estructura

Figure 15: funcionamiento: cambiarValor(Object valorViejo, Object valorNuevo)

1.4.8 void mostrar(), void mostrarClaves(), void mostrarValores()

void mostrar(). Imprime la lista completa.

void mostrarClaves(): Imprime las claves.

void mostrarValores(): Imprime los contenidos.

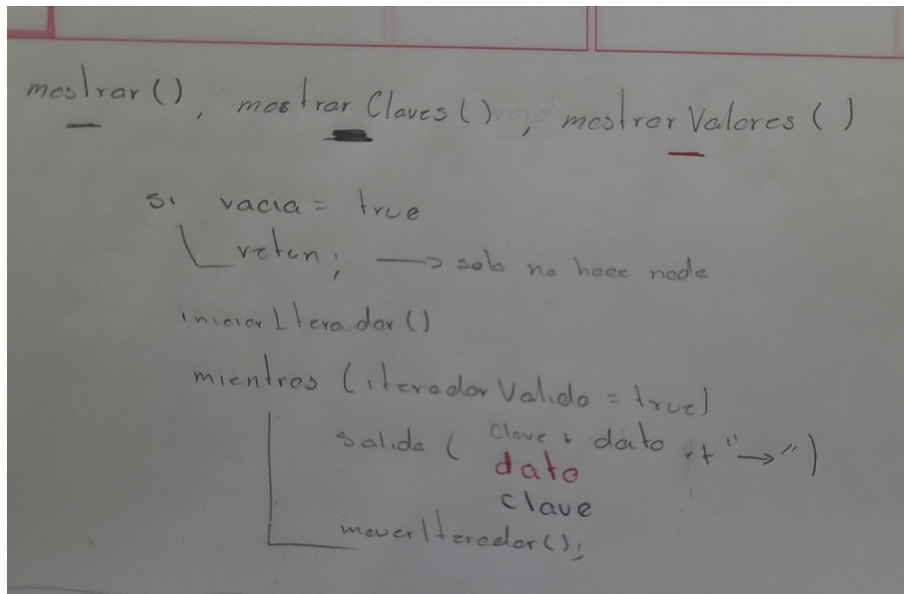


Figure 16: Análisis: void mostrar(), void mostrarClaves(), void mostrarValores()

Explicación

Para estos 3 metodos se hizo exactamente los mismo, primero se reviso que la lista no estuviera vacía, si fuera así, no hace nada, pero caso contrario imprimirá en cada métodos cosas diferentes, **Imprime la lista completa Valor:clave, mostrarClaves(): Imprime las claves y mostrarValores(): Imprime los contenidos**, así como podemos ver en la figura 17.

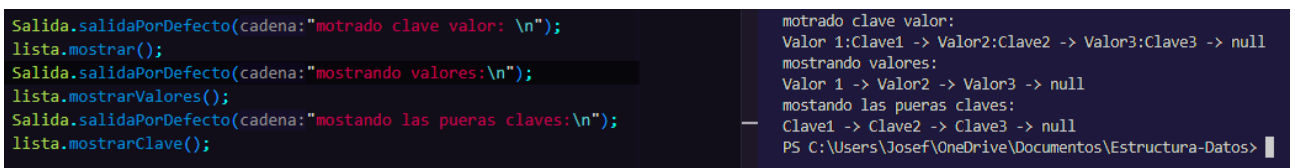


Figure 17: funcionamiento: void mostrar(), void mostrarClaves(), void mostrarValores()

1.4.9 Arreglo aListasEstaticas()

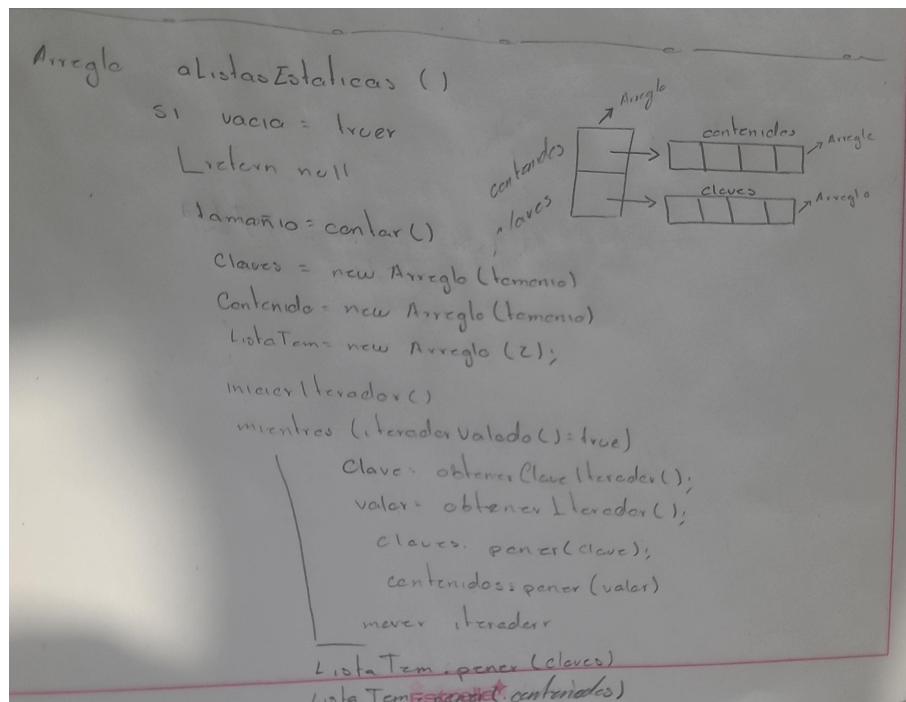


Figure 18: Análisis: aListasEstaticas()

Explicación

Para este caso lo que se hizo fue revisar de primero que la lista no estuviera sola, si fuera así regresaría null, pero si no, lo que se haría es que se crearan 3 arreglos, contenidos y claves, que tienen el tamaño de los elementos en la lista, y un tercero, listaTem que solo tiene dos espacios, ya que se itera por la lista sacando dato y clave y mandándolos a los respectivos arreglos, una vez estos arreglos se llenaran, se agregarían en la listaTem, donde en la posición 0 es para claves y la posición 1 para contenidos, y por último se regresaría este arreglo, como podemos ver en la figura 19.

```

Salida.salidaPorDefecto(cadena:"convirtiendo a lista estatica: \n");
Arreglo arre = lista.aListasEstatica();
Arreglo contenido =(Arreglo) arre.obtener(indice:1);
Arreglo claves = (Arreglo) arre.obtener(indice:0);
Salida.salidaPorDefecto(cadena:"contenido:\n");
contenido.imprimir();
Salida.salidaPorDefecto(cadena:"claves:\n");
claves.imprimir();
    
```

```

cobirtiendo a lista estatica:
contenido:
Valor 1
Valor2
Valor3
claves:
Clave1
Clave2
Clave3
PS C:\Users\Josef\OneDrive\Docu
    
```

Figure 19: funcionamiento: aListasEstaticas()

1.4.10 void vaciar()

Vacía la lista.

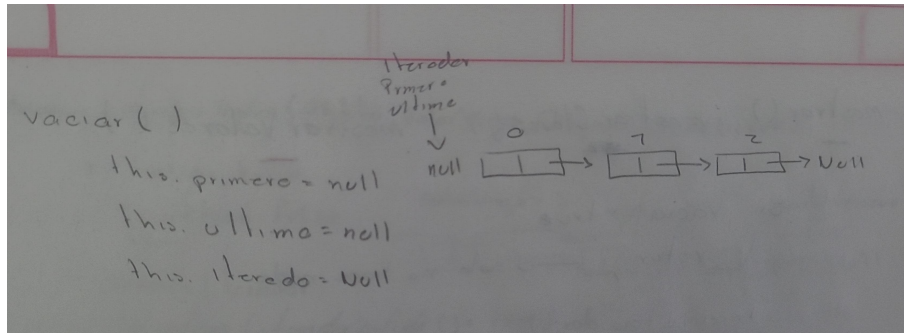


Figure 20: Análisis: vaciar()

Explicación

Para este caso solo se regreso las variables a un posición antes del primero, que en este caso seria null, así que tanto primero, ultimo y iterados tienen ese valor, dejando en el olvido los elementos que esta contenía, así como podemos ver en la figura 21.

<pre> Salida.salidaPorDefecto(cadena:"vaciendo lista:\n"); Salida.salidaPorDefecto(cadena:"contenido antes de vaciar\n"); lista.mostrar(); Salida.salidaPorDefecto(cadena:"despues de vaciar\n"); lista.vaciar(); lista.mostrar(); </pre>	<p>Claves</p> <p>vaciando lista:</p> <p>contenido antes de vaciar</p> <p>Valor 1:Clave1 -> Valor2:Clave2 -> Valor3:Clave3 -> null</p> <p>despues de vaciar</p> <p>null</p> <p>PS C:\Users\Josef\OneDrive\Documentos\Estructura-Datos></p>
---	---

Figure 21: funcionamiento: vaciar()

1.4.11 Object obtener(Object clave)

Vacía la lista.

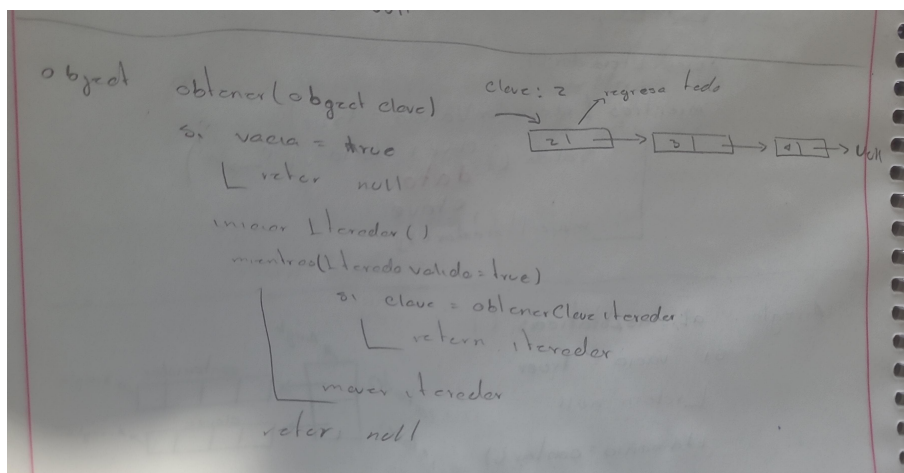


Figure 22: Análisis: obtener(Object clave)

Explicación

En este caso lo que se hizo fue revisar que no estuviera vacia la lista, si fuera asi se regresa

nullm pero caso contrario se itera sobre la lista hasta que la clave coincida con laguna existente, y si eso pasa se regresa el contenido, de no encontrarla se regresa null, como en la figura 22 y 23.

```
Salida.salidaPorDefecto(cadena:"obteniendo Clave1\n");
Salida.salidaPorDefecto(lista.obtener(clave:"Clave1")+"");
```

```
Clave3
obteniendo Clave1
Valor 1
PS C:\Users\Josef\One
```

Figure 23: funcionamiento: obtener(Object clave)

1.4.12 boolean vacia()

Indica si la lista está vacía.

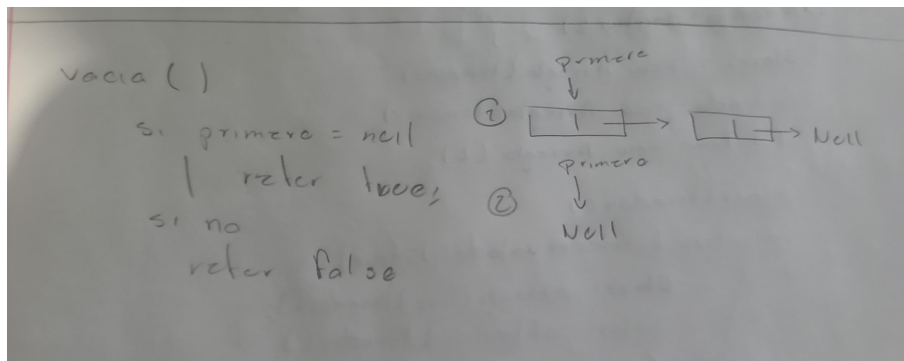


Figure 24: Análisis: vacia()

Explicación

en este caso lo que se hizo fue ver que la lista no estuviera sola, esto se ve revisando la variable primero, ya que esta si hay contenido siempre estará ocupada, y así se vería su funcionamiento como en la figura 25.

```
lista.vaciar();
if(lista.vacio()){
    Salida.salidaPorDefecto(cadena:"esta vacia la lista");
}
```

```
Clave3
obteniendo Clave1
Valor 1
esta vacia la lista
PS C:\Users\Josef\One
```

Figure 25: funcionamiento: vacia()

1.4.13 int cantidad()

Regresa el tamaño de la lista

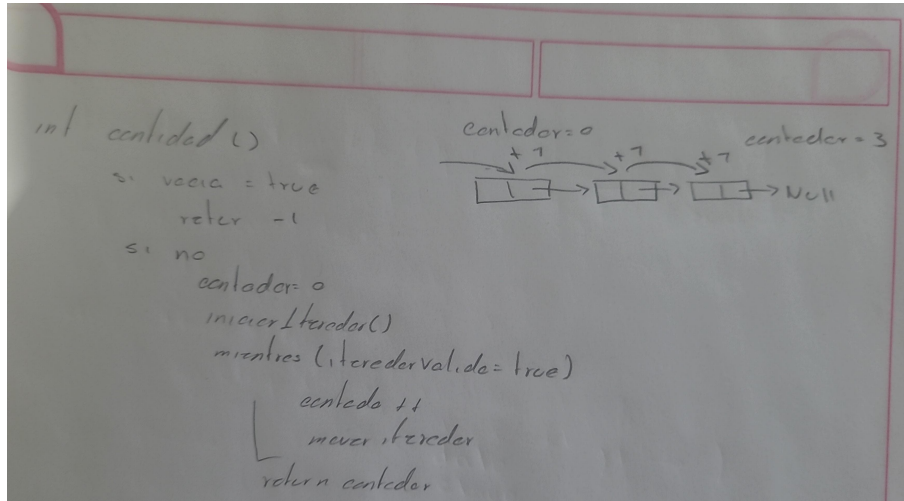


Figure 26: Análisis: cantidad()

Explicación

En este caso lo que se hizo fue ver que no estuviera vacía la lista, si lo estuviera no se hace nada, pero caso contrario se hace un contador que aumenta en 1 cada que hay un iterador valido, y cuando este se detiene regresa el contador, como en la imagen 27.

```

Salida.salidaPorDefecto(cadena:"cuantos elemento hay\n").
int canti = lista.cantidad();
if(canti > 0){
    Salida.salidaPorDefecto(canti+"\n");
}
    
```

Clave3
obteniendo Clave1
Valor 1
cuantos elemento hay
3
PS C:\Users\JOSE\One

Figure 27: funcionamiento: cantidad()

2 Código Agregado - UML

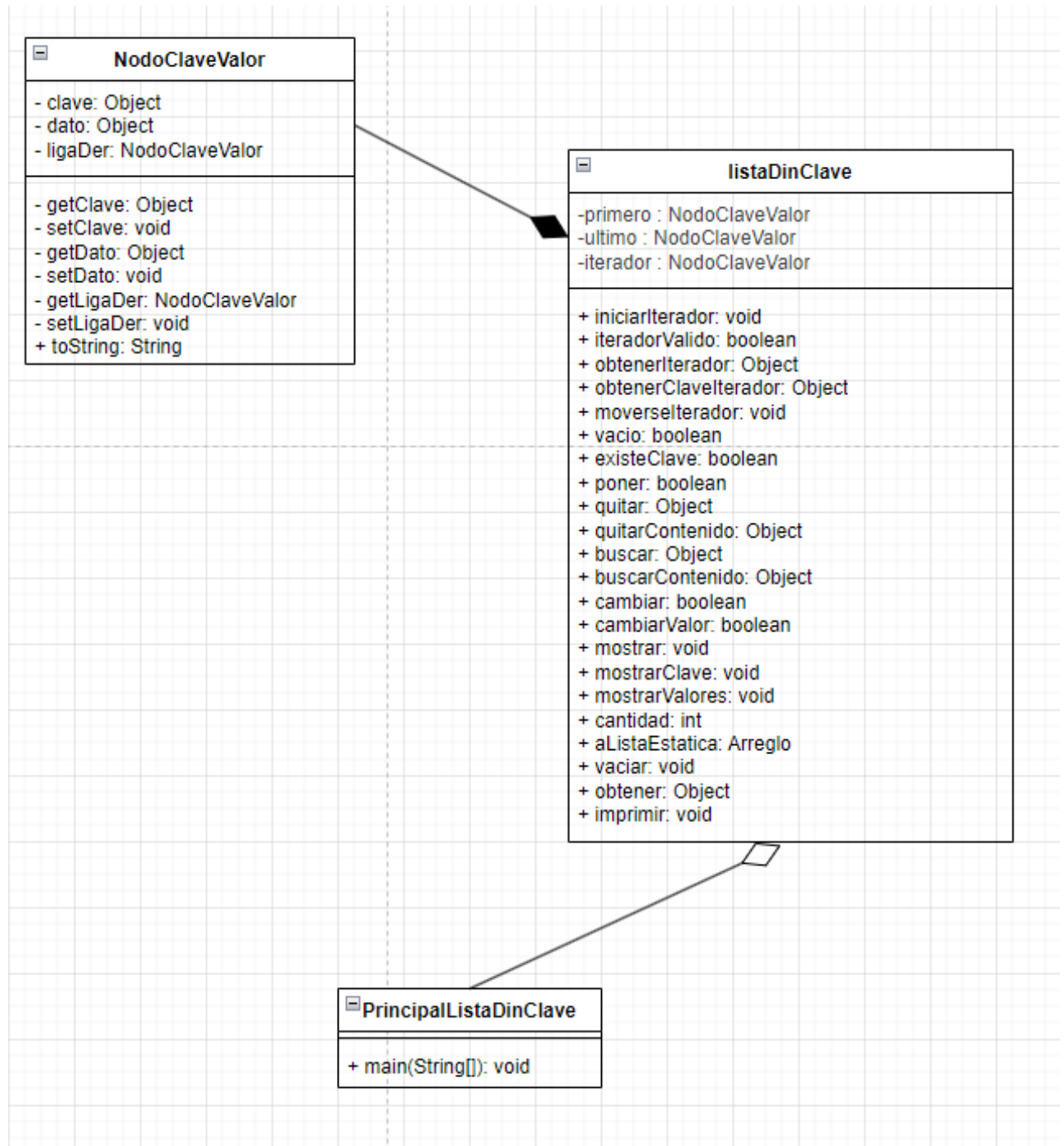


Figure 28: UML

3 Pre-evaluación del Alumno

Criterio	Evaluación
Cumple con la funcionalidad solicitada	Sí
Dispone de código auto-documentado	Sí
Dispone de código documentado a nivel de clase y método	Sí
Dispone de indentación correcta	Sí
Cumple la POO	Sí
Dispone de una forma fácil de utilizar el programa para el usuario	Sí
Dispone de un reporte con formato IDC	Sí
La información del reporte está libre de errores de ortografía	Sí
Se entregó en tiempo y forma la práctica	No
Incluye el código agregado en formato UML	Sí
Incluye las capturas de pantalla del programa funcionando	Sí
La práctica está totalmente realizada (especifique el porcentaje completado)	90%

Table 1: Evaluación de la práctica

4 Conclusión

Se vio una manera de el como usar las lista ligadas con uso de valores donde por medio de el valor podemos hacer bastantes cosas como hemos visto en esta practica

Nota: mi estado mental ya esta como esta gato.



Figure 29: yo haciendo la practica

5 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.