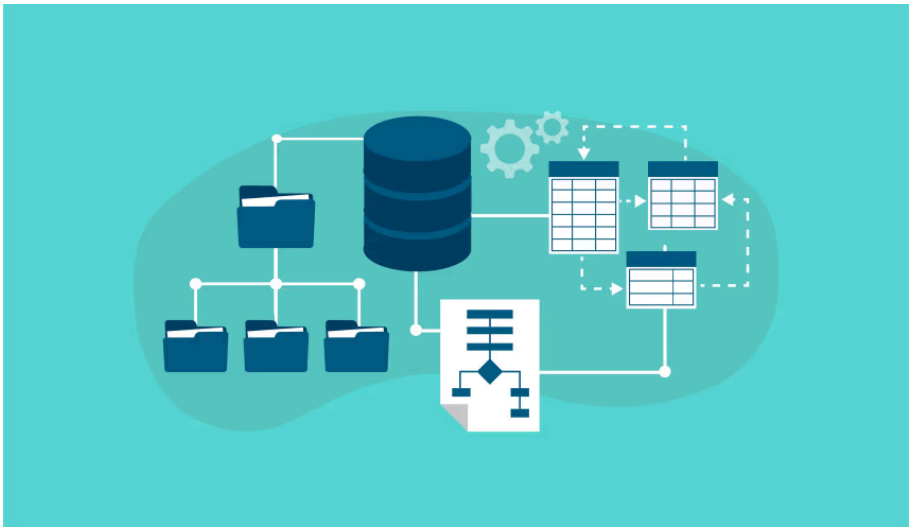
Actividad 1

Materia: Estructura de Datos

Maestro: Adalberto Emmanuel Rojas Perea

Alumna: Isabel Kyra Romero Borunda

TECMILENIO



Indíce

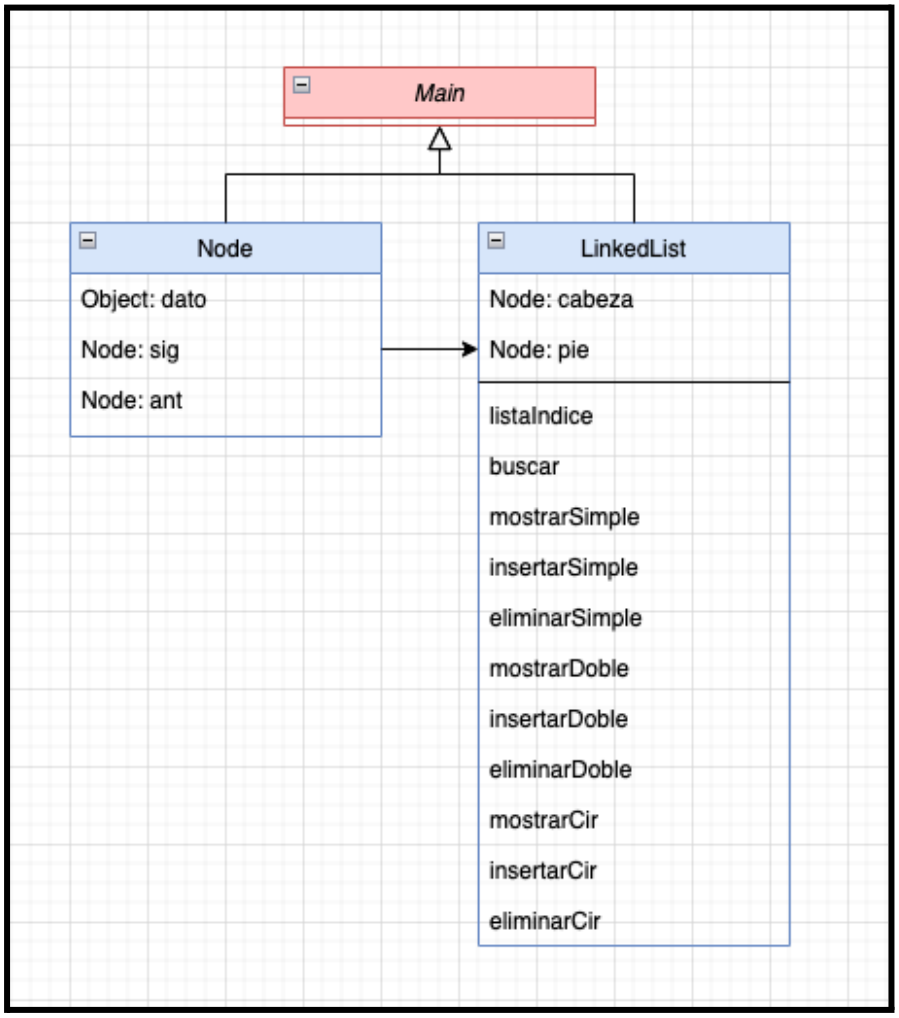
Introducción

Se describe el diseño e implementación de un programa en Java que trabaja con diferentes tipos de **listas enlazadas**: simples, dobles y circulares. Estas estructuras de datos son fundamentales en la programación, ya que permiten el manejo dinámico de información mediante nodos que se enlazan entre sí.

El objetivo del programa es demostrar cómo funcionan las operaciones básicas de inserción, eliminación y recorrido en cada tipo de lista. Se incluyen ejemplos y capturas de pantalla que muestran la ejecución del programa.

Implementación

Diagrama de Clases



Listas

Las listas se crean como objetos de LinkedList. Cada lista es de un tipo de enlace diferente(simple, doble, circular.

```
LinkedList lSimple = new LinkedList();
LinkedList lDoble = new LinkedList();
LinkedList lCircular = new LinkedList();
```

La clase LinkedList utiliza la clase de Node para crear los nodos con sus enlaces.
Esta clase cuenta con diferentes métodos para cada tipo de enlace, como se ve en el diagrama de arriba, cada uno para hacer diferentes operaciones.

```
public class LinkedList {
    Node cabeza;
    Node pie;
```

Estructura de las listas

Clase Nodo

```
public class Node<E> {
    Object dato;
    Node sig;       // Siguiente
    Node ant;       // Anterior
```

(Es mejor hacer diferentes clases de nodos dependiendo del enlace que se va a utilizar).

Lista Simple

La lista simple solo utilizaría el Nodo con un enlace, en este caso el nodo “sig”. Quedaría como en la siguiente imagen:

```
34 -> 23.543 -> manzana -> true -> null
```

Lista Doble

Esta lista utiliza dos enlaces para los nodos que no son el primero ni el último, aquí si se utilizan el los nodos “sig” y el “ant”.

```
34 <-> 34.76 <-> false <-> 34 <-> null
```

Lista Circular

La lista Circular solo utiliza el enlace “sig”. El último enlace en la imagen lleva hacia el primer dato que dice “342”.

```
342 -> 83 -> 65.8234 -> pera -> false -> 387.124 -> (circular)
```

Operaciones

Insertar

Todos los métodos se hicieron de forma que se inserte el dato después del último nodo. También los métodos son parecidos, la diferencia yace en que para unas listas se le agrega una acción más a la hora de enlazar los nodos.

Lista Simple

El método acepta un dato (*object dato*), y se crea un nuevo nodo dentro. Si la **cabeza**(primer nodo) es **null** significa que la lista está vacía, por lo cual agrega el dato como el primero de la lista.

Si la **cabeza no es null** el último nodo (pie) ahora apunta al nuevo nodo, y luego actualizamos pie para que el nuevo nodo sea el último de la lista.

```java
public void insertarSimple(Object dato) {
    Node nuevo = new Node(dato);

    // Agrega directamente el dato al final en vez de recorrer la lista
    if (cabeza == null) {
        cabeza = pie = nuevo;
    } else {
        pie.sig = nuevo;
        pie = nuevo;
    }

}
```

Lista Doble

Aquí básicamente es lo mismo, pero agregamos la acción donde el **nuevo pie** también **apunta hacía el dato anterior** además del siguiente.

```java
// Agrega directamente el dato al final en vez de recorrer la lista
if (cabeza == null) {
    cabeza = pie = nuevo;
} else {
    pie.sig = nuevo;    // el último apunta al nuevo
    nuevo.ant = pie;    // el nuevo apunta hacia atrás
    pie = nuevo;        // ahora el nuevo es el último
}
```

Lista Circular

El método acepta el pie de la lista(*Node pie*) y un dato(*object dato*). Si la lista está vacía se inserta el dato con el enlace apuntando hacia sí mismo. Y si no está vacía lo inserta en el último directamente donde el enlace del **nuevo pie apunta hacia el primer nodo**. Es como la lista simple, solo que el último dato apunta hacia el primero.

```
// Inserta el dato en una lista Circular
public Node insertarCir(Node pie, Object dato) {
    Node nuevo = new Node(dato);

    // El nodo se apunta a si mismo
    // Lista vacia
    if (pie == null) {
        nuevo.sig = nuevo;
        return nuevo;
    }

    // Inserta despues del ultimo
    nuevo.sig = pie.sig;
    pie.sig = nuevo;
    pie = nuevo; // actualiza el ultimo

    return pie;

    }
```

Eliminación

Al igual que en insertar, los métodos son parecidos, solo se diferencian a la hora de enlazar nodos.

```
public static void listaIndice(Node cabeza) {
    int index = 1; // Empezamos el índice en 1
    while (cabeza != null) {
        System.out.println(index + ": " + cabeza.dato);
        cabeza = cabeza.sig;
        index++;
    }
}
```

Se agregó un método para poder imprimir la lista con el índice de cada dato para que el usuario solo ingrese el índice del dato que desea eliminar.

Lista Simple

Los datos que recibe el método son la cabeza(*head*) de la lista y el índice del dato elegido a eliminar.
Si la cabeza está vacía nos devuelve un null. Aquí se presentan 3 casos:

- **Caso 1:** La cabeza es eliminada
El siguiente nodo se guarda como la cabeza.
- Caso 2: El nodo está entre la cabeza y el último nodo
Busca el índice del nodo y hace que el nodo anterior sea el actual, luego enlaza el nodo actual con el siguiente.
- Caso 3: El último es eliminado

```
public Node eliminarSimple(Node head, int index) {
    if(head == null)
        return null;

    Node temp = head;   //Guarda la referencia a la cabeza
    Node prev = null;

    // Si la cabeza se elimina
    if (index == 1) {
        head = temp.sig;
        return head;
    }

    // Si el nodo este en medio
    for (int i = 1; temp != null && i < index; i++) {
        prev = temp;
        temp = temp.sig;
    }

    // Elimina el nodo
    if (temp != null) {
        prev.sig = temp.sig;
    }
    else {
        System.out.println("No se encuantra el dato");
    }

    return head;
}
```

Lista Doble

```java
public Node eliminarDoble(Node head, int index) {
    if (head == null) {
        return head;
    }

    Node temp = head;

    // Va hacia el nodo seleccionado
    for (int i = 1; temp != null && i < index; ++i) {
        temp = temp.sig;
    }

    if (temp == null) {
        return head;
    }

    // Actualiza el siguiente enlace del nodo anterior
    if (temp.ant != null) {
        temp.ant.sig = temp.sig;
    }

    // Actualiza el anterior enlace del siguiente nodo
    if (temp.sig != null) {
        temp.sig.ant = temp.ant;
    }

    // Si el pie quiere ser eliminado
    if (temp.sig == null) {
        pie = temp.ant;
    }

    // Si la cabeza quiere ser eliminada
    if (head == temp) {
        head = temp.sig;
    }

    return head;
```

Lista Circular

```java
public Node insertarCir(Node pie, Object dato) {
    Node nuevo = new Node(dato);

    // El nodo se apunta a si mismo
    // Lista vacia
    if (pie == null) {
        nuevo.sig = nuevo;
        return nuevo;
    }

    // Inserta despues del ultimo
    nuevo.sig = pie.sig;
    pie.sig = nuevo;
    pie = nuevo; // actualiza el ultimo

    return pie;

    }
```

Mostrar

Aquí como en las demás operaciones, los métodos siguen la misma lógica solo que con unas pequeñas modificaciones con diferentes listas.

Lista Simple

Empieza desde la cabeza y utiliza un ciclo while donde mientras sea diferente a null va a seguir imprimiendo los datos. Utiliza una flecha “->” para mostrar hacia donde apunta el enlace de cada nodo.

```
public void mostrarSimple() {
    Node actual = cabeza;
    while (actual != null) {
        System.out.print(actual.dato + " -> ");
        actual = actual.sig;
    }
    System.out.println("null");

}
```

Lista Doble

Aquí es lo mismo, no se detiene hasta llegar a un null y la flecha apunta hacia ambos lados “<->”.

```
public void mostrarDoble() {
    Node actual = cabeza;
    while (actual != null) {
        System.out.print(actual.dato + " <-> ");
        actual = actual.sig;
    }
    System.out.println("null");

}
```

Lista Circular

Aquí es diferente debido a que esta lista no termina en null, utiliza el nodo que sigue del **last** para ubicar la cabeza de la lista. Mientras el dato en el que se encuentra sea diferente de la cabeza va a seguir imprimiendo los datos. La flecha que utiliza es la misma que en la lista simple “->”.

```java
public static void mostrarCir(Node last) {
    if (last == null) {
        System.out.println("Lista vacía");
        return;
    }
    Node temp = last.sig; // cabeza
    do {
        System.out.print(temp.dato + " -> ");
        temp = temp.sig;
    } while (temp != last.sig);
    System.out.println("(circular)");
}
```

Buscar

Este método fue creado para todas las listas, no funcionó como se quería. Solo podía encontrar datos que fueran de tipo String(se va a ver más adelante en los resultados).

```java
static boolean buscar(Node cabeza, Object dato) {
    Node temp = cabeza;

    while (temp != null) {

        if (temp.dato != null && temp.dato.equals(dato))
            return true;

        // Se mueve al nodo que sigue
        temp = temp.sig;
    }
     return false;
}
```

Tipos de Datos

Se manejan tipos de datos primitivos como el int, float, string, char y boolean. En la clase de “Node” se utilizó “Object” para que soporte todos estos tipos.

```java
public class Node {
    Object dato;
    Node sig;       // Siguiente
    Node ant;       // Anterior
```

Luego ya en el menú de la listas si se elige insertar un elemento te pregunta qué tipo de dato deseas agregar, la opción elegida es como se guarda el dato. En las imágenes siguientes se muestran 2 ejemplos, el primero donde se inserta un dato de tipo **int** y la segunda donde es de tipo float.

Imagen 1

```
Que desea hacer?
1-.Insertar 2-.Eliminar 3-.Buscar dato 4-.Imprimir Lista 5-.Regresar
1

Que tipo de dato desea insertar?
1-.Entero 2-.Float 3-.String 4-.Char 5-.Boleano
1

Inserte el dato(int): 34
```

Imagen 2

```
Que desea hacer?
1-.Insertar 2-.Eliminar 3-.Buscar dato 4-.Imprimir Lista 5-.Regresar
1

Que tipo de dato desea insertar?
1-.Entero 2-.Float 3-.String 4-.Char 5-.Boleano
2

Inserte el dato(float): 34.76
```

Resultados

Lista Simple

Se elige el tipo de lista en la que se opera en este caso Simple.

```
Qué tipo de lista desea utilizar?(inserte un número)
1-.Simple 2-.Doble 3-.Circular 4-.Ver Listas 5-.Demo 6-.Salir
1
```

Se elige la operación a hacer en la lista, donde se eligió insertar un dato.

```
Que desea hacer?
1-.Insertar 2-.Eliminar 3-.Buscar dato 4-.Imprimir Lista 5-.Regresar
1

Que tipo de dato desea insertar?
1-.Entero 2-.Float 3-.String 4-.Char 5-.Boleano
1

Inserte el dato(int): 34
```

Impresión de la lista:

```
Que desea hacer?
1-.Insertar 2-.Eliminar 3-.Buscar dato 4-.Imprimir Lista 5-.Regresar
4
34 -> 23.543 -> manzana -> true -> null
```

Eliminación de un dato:

```
Que desea hacer?
1-.Insertar 2-.Eliminar 3-.Buscar dato 4-.Imprimir Lista 5-.Regresar
2

Digite el index del dato que desea eliminar
1: 34
2: 23.543
3: manzana
4: true
4
Lista actulizada:

34 -> 23.543 -> manzana -> null
```

Encontrar un dato en la lista, aquí me di cuenta de que sólo se pueden encontrar Strings.

```
Que desea hacer?
1-.Insertar 2-.Eliminar 3-.Buscar dato 4-.Imprimir Lista 5-.Regresar
3

Digite el dato que desea buscar
manzana
Si está

Que desea hacer?
1-.Insertar 2-.Eliminar 3-.Buscar dato 4-.Imprimir Lista 5-.Regresar
3

Digite el dato que desea buscar
34
No está
```

Lista Doble

Se elige la lista Doble.

```
Qué tipo de lista desea utilizar?(inserte un número)
1-.Simple 2-.Doble 3-.Circular 4-.Ver Listas 5-.Demo 6-.Salir
2
```

Se insertan datos.

```
Que desea hacer?
1-.Insertar 2-.Eliminar 3-.Buscar dato 4-.Imprimir Lista 5-.Regresar
1

Que tipo de dato desea insertar?
1-.Entero 2-.Float 3-.String 4-.Char 5-.Boleano
1

Inserte el dato(int): 34
```

Impresión de la lista:

```
Que desea hacer?
1-.Insertar 2-.Eliminar 3-.Buscar dato 4-.Imprimir Lista 5-.Regresar
4
34 <-> 34.76 <-> false <-> 34 <-> null
```

Eliminación de un dato. Hubo fallos en el código, no salió como se esperaba.

```
Que desea hacer?
1-.Insertar 2-.Eliminar 3-.Buscar dato 4-.Imprimir Lista 5-.Regresar
2

Digite el index del dato que desea eliminar
1: 34
2: 34.76
3: false
4: 34
1
Lista actulizada:

34 <-> 34.76 <-> false <-> 34 <-> null
```

Lista Circular

Selección de la lista Circular.

```
Qué tipo de lista desea utilizar?(inserte un número)
1-.Simple 2-.Doble 3-.Circular 4-.Ver Listas 5-.Demo 6-.Salir
3
```

Inserción de datos.

```
Que desea hacer?
1-.Insertar 2-.Eliminar 3-.Buscar dato 4-.Imprimir Lista 5-.Regresar
1

Que tipo de dato desea insertar?
1-.Entero 2-.Float 3-.String 4-.Char 5-.Boleano
45
```

Impresión de lista.

```
Que desea hacer?
1-.Insertar 2-.Eliminar 3-.Buscar dato 4-.Imprimir Lista 5-.Regresar
4
43.34 -> per -> K -> (circular)
```

Eliminación de datos. Se puede ver que hubo fallos en el código, no salió como se esperaba.

```
Digite el index del dato que desea eliminar
1: K
2: 43.34
3: per
4: K
5: 43.34
6: per
7: K
8: 43.34
```

Conclusión

Me apoye de la siguiente página para crear el código: https://www.geeksforgeeks.org/dsa/linked-list-data-structure

Se puede ver que hubo fallas en el programa, ya que no funcionaron los métodos de eliminación de la lista Doble ni el método de búsqueda de elementos.
El método de búsqueda sólo detecta Strings y no otros tipos de datos.
La lista Circular fue la que más fallas tuvo debido a que los métodos de eliminación e impresión del menú principal no funcionaron.