INSTITUTO POLITECNICO NACIONAL

ESCUELA SUPERIOR DE COMPUTO

INGENIERIA EN INTELIGENCIA ARTIFICIAL

PARADIGMAS DE PROGRAMACION

PRACTICA NUMERO 2

RECURSION EN PROGRAMACION FUNCIONAL

VAZQUEZ PEREZ ANGEL MARTIN

GRUPO 3BV2

INTRODUCCION

En esta práctica estaremos implementando una lista simplemente enlazada de manera recursiva, tomaremos la base de la lista funcional y la convertiremos en la cual las funciones sean todas recursivas y lo más importante que sigan el paradigma funcional.

Aunque Java incluye ya una librería donde esta implementada la lista simplemente enlazada y tiene muchas funciones implementadas para que puedas usarla durante el código del programa

En la práctica anterior implementamos funciones que seguían el paradigma funcional, había funciones de primer orden y hasta de segundo orden, durante esta práctica igualmente habrá funciones que sean de primer orden y algunas otras de segunda orden

En la lista que estaremos implementando el nodo tendrá en su contenido el nodo siguiente, y un string, este string e muy importante porque no va a poder almacenar otro tipo de datos más que string, si tú vas a querer ingresar un entero no te va a dejar porque la lista esta implementada para que solo reciba string.

Puede que la implementación no quede del todo bonita, pero hace su función, en el main vamos a recibir todo lo que programemos de las funciones, habrá funciones que sean de tipo void las cuales solo realizaran algunos algoritmos como l oes el de ordenamiento, pero mientras tanto en los métodos de eliminar y de buscar un nodo con una información, en el main vamos a recibir el nodo eliminado o si el nodo se busco y se va a llamar a una función para que me saque en pantalla lo ocurrido con el nodo.

Esto anteriormente mencionado no puede ir dentro de las funciones recursivas funcionales, ya que siguiendo el método del paradigma funcional ,en este no se puede introducir ni leer cosas del teclado ,debe trabajar independiente, con los datos que este contenga, igualmente tampoco puede sacar a pantalla algo, o llamara a una funciona para que me muestre en mi pantalla algo, esto lo va a trabajar el main

MENU DE NUESTRO PROGRAMA

```
public class Menu {
    public static void visualizarMenu() {
        System.out.println("ESTE PROGRAMA HACE TODAS LAS FUNCIONE DE FORMA RECURSIVA");
        System.out.println( "Menu  Que operacion quieres realizar\n" );
        System.out.println( "  1. Crear al inicio.");
        System.out.println( "  2. Crear al final.");
        System.out.println( "  3. Imprimir lista recursivamente.");
        System.out.println( "  4. Insertar al inicio.");
        System.out.println( "  5. Insertar al final.");
        System.out.println( "  6. Insertar antes de un nodo X.");
        System.out.println( "  7. Insertar despues de un nodo X.");
        System.out.println( "  8. Eliminar al inicio.");
        System.out.println( " 9. Eliminar ultimo.");
        System.out.println( " 10. Eliminar nodo con informacion X.");
        System.out.println( " 11. Eliminar nodo anterior con informacion X.");
        System.out.println( " 12. ordenar la lista.");
        System.out.println( " 13. Busqueda de un elemento.");
        System.out.println( " 14. Salir del programa.");
        System.out.print( " Elija una opcion ... ");
    }
```

Todas estas opciones tenemos en nuestro Código y cada una de ellas está de forma recursiva y siguiendo el paradigma funcional.
El usuario va a escoger que opción realizar, por lógica debe de seleccionar primero el caso 1 o 2 para crear la lista.

EL usuario va a introducir la cantidad de nombres que quiera ingresar y va a escribir todos los nombres a ingresar, y a mis funciones va a llegar un arreglo de nombres para insertarlos en nuestra lista como lo muestra la siguiente imagen

```
public static void crearAlInicio(String[] dato, int n) {
        p.nombre = dato[n-1];
        p.siguiente = null;
        insertarInicio(dato,n-1);
    }

public static void crearAlFinal(String[] dato, int n) {
        p.nombre = dato[n-1];
        p.siguiente = null;
        insertarFinal(p,dato,n-1);
    }
```

Funciones de crear la lista recibiendo un arreglo de string y llama a la función recursiva llamada insertarinicio o insertarfinal según el caso;

FUNCIONES INSERTAR

INSERTAR INICIO DE LA LISTA

Estas funciones siguen el paradigma funcional y recibe el arreglo de nombres que el usuario va a ingresar, funciona de la siguiente manera.

```
public static void insertarInicio(String[] dato, int n) {
    if(n==0){
        return;
    }
    Nodo q = new Nodo();
    q.nombre = dato[n-1];
    q.siguiente = p;
    p = q;
    insertarInicio(dato ,n-1) ;
    }
```

Función insertarinicio funciona de manera recursiva, recibe el arreglo y el número de nombres a ingresar, entra al primer condicional para ver si sigue con el funcionamiento, si el numero sigue sin ser 0, va a insertar al inicio de la lista haciendo que ese nuevo nodo sea la cabecera y que apunte al que era la cabecera, y después llama de nuevo a insertar inicio, pero disminuye el valor de n, y así funcionara hasta que n sea 0 y se salga.

INSERTAR FINAL DE LA LISTA

```
public static void insertarFinal(Nodo t, String[] dato,int  n) {
    if(n==0){
        return;
    }else{
    if (t.siguiente == null) {
        Nodo q = new Nodo();
        q.nombre = dato[n-1];
        q.siguiente = null;
        t.siguiente = q;
        insertarFinal(q,dato,n-1);
    }else{
    insertarFinal(t.siguiente, dato, n);
    }
    }
}
```

Esta función tiene otro condicional adicional, hace que mi llamada recursiva haga que llegue hasta el final de nuestra lista, lo hace con la condicional si el nodo siguiente es NULL, es indica que ya llego al final, y llegando al final ya puede empezar a ingresar todos los datos y disminuyendo el valor de n para que este sea 0;

INSERTAR ANTES DE UN VALOR X

```
public static void insertarAntesX(Nodo c,String dato, String X,Nodo a) {
    if(c==null){
        return;
    }
    if(c.nombre.equals(X) && a==null){
        String[] datos = new String[1];
        datos[0]=dato;
        insertarInicio(datos,1);
        return;
    }
    if(c.nombre.equals(X) ) {
        Nodo nuevo= new Nodo();
        nuevo.nombre=dato;
        nuevo.siguiente=c;
        a.siguiente=nuevo;
        return;
        }
    else {
        insertarAntesX(c.siguiente,dato,X,c);
        }

        }
```

Esta función recibe el nodo cabecera, el nombre a insertar, el nombre a buscar para insertar antes de él y el nodo anterior, entonces primeramente revisa si el nombre para ingresar antes de él es la cabecera, entonces llama la función de insertar inicio; si esto no sucede entonces revisa si el nombre a buscar está en esa posición del nodo, si no lo hace, se llama a si mismo pero enviándole el nodo siguiente, y el nodo anterior seria el ya revisado, cuando lo encuentre lo inserta antes que este con la ayuda del nodo anterior;

INSERTAR DESPUE DE UN X

```
public static void insertarDespuesX(Nodo c,String dato, String X) {
    if(c.nombre.equals(X)){
        Nodo nuevo=new Nodo();
        nuevo.siguiente=c.siguiente;
        nuevo.nombre=dato;
        c.siguiente=nuevo;
    }
    else if(c.siguiente==null){
            return;
    }
    else{
        insertarDespuesX(c.siguiente, dato,X);
    }
}
```

Esta función es parecida a la función anterior solo que nos necesita de tener de respaldo a el nodo anterior, igualmente hace la función recursiva buscando el nodo que contiene el nombre con el nodo siguiente, cuando ya llega hace la inserción después de ese y si no lo encuentra regresa sin éxito alguno;

FUNCIONES ELIMINAR

FUNCION ELIMINAR INICIO

```
public static Nodo eliminarInicio() {
    Nodo q = p;
    p = q.siguiente;
    return q;
}
```

Esta función retorna un nodo, y simplemente hace que el nodo que sea la cabecera de la lista sea su nodo siguiente, y retorna el primer nodo para que en el main lo reciba y lo elimina;

FUNCION ELIMINAR FINAL

```
public static Nodo eliminarUltimo(Nodo c,Nodo a) {
    if(c==null){
        Nodo t=null;
        return t;
    }
    if(c.siguiente==null){
        if (a==null){
            p=null;
        }else{
            a.siguiente=null;
        }
        return c;
    }
        Nodo x;
        x=eliminarUltimo(c.siguiente,c);
           return x;
}
```

Esta función retorna un nodo , y recibe 2 nodos, un no do y su nodo anterior después revisa si la lista está vacía, si sucede me retorna un nulo, si el valor siguiente del nodo es nulo, entonces es y final de la lista, y si sucede esto revisa si el nodo anterior es nulo entonces la lista quedara vacía, si no sucede esto, el nodo anterior va a apuntar a nulo y se retorna el nodo ultimo para que en el main se elimine, la función recursiva esta para llegar hasta el final de la lista.

ELIMINAR UN NODO CON CONTENIDO X

```
public static Nodo eliminarX(Nodo c,String X,Nodo a) {
    if(c==null){
        return null;
    }
    if(c.nombre.equals(X)){
        if(a==null){
            return eliminarInicio();
        }else if(c.siguiente==null){
            return eliminarUltimo(p,null);
        }else{
            a.siguiente=c.siguiente;
            c.siguiente=null;
            return c;
        }
    }
    else{
        Nodo Y=eliminarX(c.siguiente, X, c);
        return Y;
    }
}
```

Esta función igualmente me regresa un nodo, y recibe el string a eliminar y un nodo y su nodo anterior de respaldo, la función se hace recursiva mientras el nombre no lo encuentre en la lista, cuando lo encuentre entonces hace la función de que los nodos se apunten entre sí y regrese el nodo eliminado para que en el main se borre.

ELIMINAR NODO ANTES DE UN NODO CON INFORMACION X

```
public static Nodo eliminarAntesX(Nodo c,String X,Nodo a) {
    if(c==null){
        return null;
    }else{
    if(c.nombre.equals(X)){
        if(a==null){
            return eliminarInicio();
        }else {
            return eliminarX(p,a.nombre,null);
        }
    }
    else{
        Nodo Y=eliminarAntesX(c.siguiente, X, c);
        return Y;
    }
    }
}
```

La función hace lo mismo que la función anterior solo que es una función que llama a la función anterior pero igualmente es recursiva cuando encuentre el nodo a eliminar antes de ese, llama la función anterior para que esta se haga cargo de lo que resta.

BUSCAR NODO CON INFORMACION X

```
public static Nodo buscar(Nodo c,String X) {
    if(c==null){
        return null ;
    }else{

        if(c.nombre.equals(X)){
            return c ;
        }
        else{
            return buscar(c.siguiente,X);
        }

    }
}
```

Esta función hace la búsqueda de manera secuencial y recursiva, si la información del nodo es igual a nodo a buscar me retorna el nodo para que en el main se muestra su información, si no se encuentra me retorna un NULL;

FUNCION ORDENAR

```
    public static void ordenarBurbujaRecursivo(Nodo c) {
    if (c == null || c.siguiente == null) {
        return;
    }

    if (burbuja(c)==true) {
        ordenarBurbujaRecursivo(c);
    }
}
public static  boolean burbuja(Nodo c) {
    boolean cambio = false;
    Nodo temp = c;
    while (temp != null && temp.siguiente != null) {
        if (temp.nombre.compareTo(temp.siguiente.nombre) > 0) {
            String tmp = temp.nombre;
            temp.nombre = temp.siguiente.nombre;
            temp.siguiente.nombre = tmp;
            cambio = true;
        }
        temp = temp.siguiente;
    }
    return cambio;
}
```

Aquí para ordenar nuestra lista ocupe el método de ordenamiento de burbuja ,estuve investigando y este método a pesar de no ser el mas eficiente de todos fue el más fácil para implementar recursivamente, mi método que llamo desde el main es el método que se encuentra en la parte de arriba, el void, ya que no me va a retornar nada, esto lo primero que hace es comparar si la cabecera es NULL o el nodo siguiente de la cabecera es NULL, automáticamente si sucede algo de esto va a retornar la función, si esto no sucede entra en la condición donde se ocupaba la función burbuja la cual me va a retornar un valor booleano, dentro de esta función se va a revisar si el nombre del nodo siguiente es menor o mayor a 0 para revisar si se ejecuta el cambio de información y así sucesivamente hasta que el cambio sea falso y me diga que ya no se realizó ningún cambio y eso indica que la lista ya se encuentre ordenada.

CONCLUSIONES

En la practica pudimos implementar o hacer el cambio de las funciones que teníamos como base que seguían el paradigma funcional , de una manera muy particular y que nos ayuda a adentrarnos más en la recursividad.

Esto tiene muchas ventajas y desventajas, luego ola recursividad me hace que el gasto de memoria sea mucho mayor, pero quizá mi tiempo de ejecución sea menor, igual lo que esta practica nos ha enseñado es

Una de las lecciones más de esta práctica como abordamos problemas de manera modular y escalable. Al dividir un problema de manera recursiva. La implementación de una lista simplemente enlazada de manera recursiva, siguiendo el paradigma funcional. Nos ha brindado una comprensión más profunda de los conceptos fundamentales de la programación y nos ha equipado con herramientas poderosas para abordar desafíos futuros en el mundo de la programación en Java siguiendo lo funcional.

Esto en un futuro nos podría ayudar a conocer y convertir funciones de manera funcionalmente recursiva.