Universidad Autónoma de Zacatecas

Unidad Académica de Ingeniería Eléctrica

Programa Académico de Ingeniería de Software.

Nombre de la Práctica Recursión.

Numero de Práctica 23

Nombre de la carrera Ingeniería de Software

Nombre de la materia Lab. Estructuras de Datos

Nombre del alumno Jesús Manuel Juárez Pasillas

Nombre del docente Aldonso Becerra Sánchez

Fecha: 14/10/2021

Práctica 23: Recursión.

Introducción:

Los métodos recursivos suelen ser demasiado pequeños, pero siempre ocupan algún parámetro para poder funcionar, ya que sin este es muy difícil definir el caso base con el que para la recursión. Es por esto que en muchos de los casos se necesitara al menos dos métodos para hacer lo que antes estaba en solo un método.

Desarrollo:

Para esta práctica se pidió que se agregaran métodos recursivos a las clases ArregloDatos y ListaEncadenada. Los métodos que se piden son aquellos que ya están definidos en estas clases, pero cómo funcionan con ciclos, pueden utilizar la recursión, por lo que harán lo mismo, pero con recursión.

En muchos de los métodos que se estarán haciendo se tendrán que hacer 2 métodos para que funcione, uno que sea el base, con los mismos parámetros que el método que no utiliza la recursión y mande llamar al otro método que será el método recursivo y por lo tanto cambiaran sus parámetros. El segundo método siempre será privado para que no se pierda la integridad de los datos y solo pueda ser llamado por el primer método.

Los métodos que se vallan agregar y por lo tanto tendrán recursión se diferenciarán del resto ya que al final de su nombre tendrán un “RR” para indicar que son los métodos recursivos.

Primero se agregarán los métodos a la clase ArregloDatos, además que se pidió agregar un nuevo método el cual convierte el arreglo en una lista encadenada. Los métodos agregados son:

- Método que llama al método con la posición.
  ```
  public void imprimirRR()
  ```
  Imprime el arreglo en forma natural usando recursión.
  ```
  private void imprimirRR(int posicion)
  ```
- Método que llama al método con la posición.
  ```
  public void imprimirOrdenInversoRR()
  ```
  Imprime el arreglo en forma natrual usando recursión.
  ```
  private void imprimirOrdenInversoRR(int posicion)
  ```
- Manda llamar el método buscarRR.
  ```
  public Object buscarRR(Object elemento)
  ```

Busca un elemento en la lista usando recursión.
private Object buscarRR(Object elemento, int posicion)

- Llama al método eliminar.
public Object eliminarRR(Object elemento)

Elimina el dato pasado como parámetro.
private Object eliminarRR(Object elemento, int posicion)

- Vacía la lista haciendo uso de la recursividad.
public void vaciarRR()

- Agrega datos de una lista a la lista actual.
public boolean agregarListaRR(Object listaDatos2)

- Llama al método invertirRR.
public void invertirRR()

Invierte el arreglo usando recursión.
private void invertirRR(ArregloDatos tmp)

- Llama al método contarRR.
public int contarRR(Object elemento)

Cuenta el número de veces que aparece el elemento dado utilizando recursión.
private int contarRR(Object elemento, int posicion)

- Rellena el arreglo con un dato duplicado la cantidad de veces indicada.
public void rellenarRR(Object elemento, int cantidad)

- Llama al método clonarRR.
public Object clonarRR()

Clona el arreglo usando recursividad.
private Object clonarRR(int posicion,ArregloDatos arr)

- Rellena el arreglo con el elemento dado, utilizado recursión.
public void rellenarRR(Object elemento)

- Llama al método arregloAListaRR.
public ListaEncadenada arregloAListaRR()

Agrega los datos del arreglo a una lista enlazada.
private ListaEncadenada arregloAListaRR(int posicion,ListaEncadenada lista)

Como se puede observar mas de la mitad de métodos tuvieron que ser dobles por lo que a la hora de utilizar la recursión se tiene que evaluar esto muy bien ya que sin esto se pueden cometer errores.

En el caso de las listas el numero de métodos dobles será mucho mayor ya que para estas es muy importante saber en qué nodo se encuentra actualmente en la lista, por lo que se muchos métodos serán dobles, aunque hay otros métodos los cuales no es necesario hacer dos métodos debido a su estructura de los parámetros y a lo que hacen.

En la clase ListaEncadenada se agregaron los siguientes metodos:

- Manada llamar el método imprimirRR con el nodo frente.
```
public void imprimirRR(){

Imprime la lista.
private void imprimirRR(Nodo tmp)
```

- Manada llamar el método imprimir con el nodo frente.
```
public void imprimirOrdenInversoRR()

Imprime la lista en orden inverso utilizando recursión.
private void imprimirOrdenInversoRR(Nodo tmp)
```

- Manda llamar el método buscarRR con los parámetros de info, nodo y la posición.
```
public Object buscarRR(Object elemento)

Busca un elemento en la lista usando recursión.
private Object buscarRR(Object elemento, Nodo tmp, int posicion)
```

- Manada llamar al método eliminarFinalRR y le manda el nodo frente.
```
public Object eliminarFinalRR()

Elimina el último dato de la lista.
private Object eliminarFinalRR(Nodo tmp)
```

- Llama al método obtenerRR.
```
public boolean cambiarRR(int indice,Object elemento)

Busca el elemento que se encuentre en el índice indicado.
private boolean cambiarRR(int indice,Object elemento,Nodo tmp, int cont)
```

- Llama al método obtenerRR.
```
public Object obtenerRR(int indice)

Busca el elemento que se encuentre en el índice indicado.
private Object obtenerRR(int indice,Nodo tmp, int cont)
```

- Llama al método esIgualRR.
  ```
  public boolean esIgualRR(Object listaDatos2)

  Evalúa si las dos listas son iguales recorriendo sus nodos.
  private boolean esIgualRR(Nodo n1, Nodo n2)
  ```

- Llama al método correspondiente para agregar la lista dependiendo si es para una lista o un arreglo.
  ```
  public boolean agregarListaRR(Object listaDatos2)

  Agrega los datos de la lista a la lista actual.
  private boolean agregarListaRR(Nodo nodo)

  Agrega los datos del arreglo a la lista actual.
  private boolean agregarListaRR(ArregloDatos arr,int posicion)
  ```

- Manda llamar al método invertirRR y le asigna los valores de la lista que regresa a la actual.
  ```
  public void invertirRR()

  Invierte la lista actual.
  private ListaEncadenada invertirRR(ListaEncadenada lista)
  ```

- Llama al método contarRR.
  ```
  public int contarRR(Object elemento)

  Cuenta el número de apariciones del elemento en la lista.
  private int contarRR(Object elemento, int contador, Nodo tmp)
  ```

- Agrega el elemento el número indicado con cantidad.
  ```
  public void rellenarRR(Object elemento, int cantidad)
  ```

- Llama al método clonarRR.
  ```
  public Object clonarRR()

  Agrega los elementos de la lista actual a la lista.
  private ListaEncadenada clonarRR(Nodo tmp, ListaEncadenada lista)
  ```

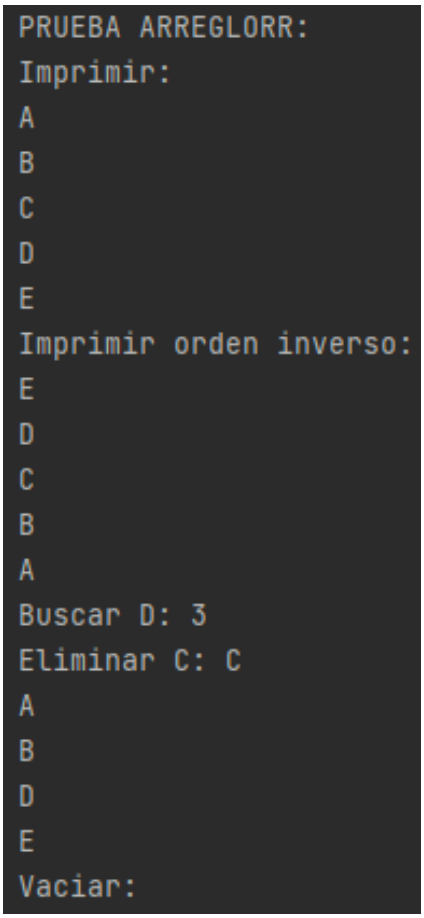
En este caso solamente un método se pudo hacer solo, ya que todos los demás necesitaron de dos métodos, además que hubo un caso especial el cual necesito 3 métodos, 1 base y 2 recursivos, ya que tenía dos escenarios diferentes los cuales se necesitaban hacer recursivos

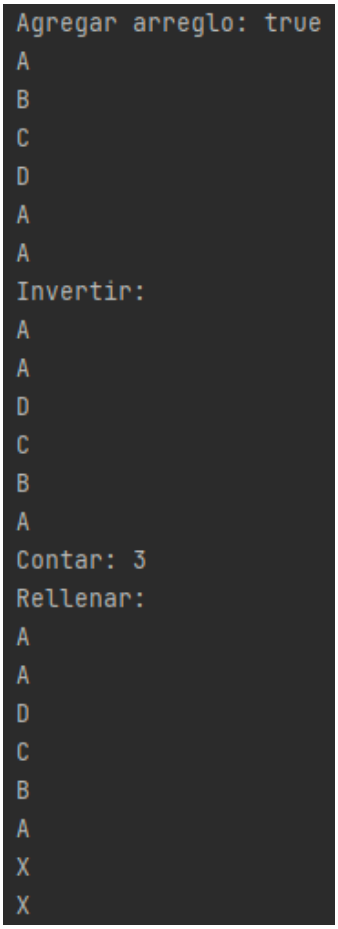
Nota: Toda la documentación del proyecto esta agregada en la carpeta “doc” dentro de la carpeta del proyecto (“edylab_2021_23/doc”).

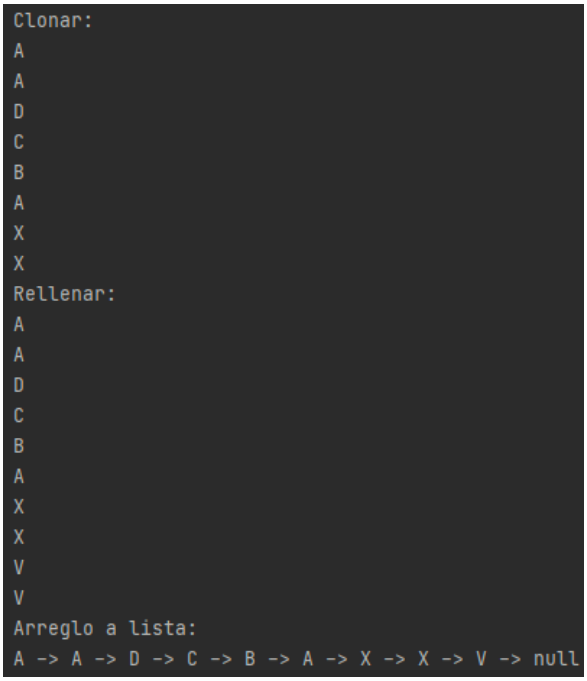
Capturas del programa funcionando:

La prueba de todos los metodos echos en esta practica se hicieron en una sola clase, primero se probaron los metodos agregados al arreglo, luego los metodos agregados a la lista.

PruebaArregloListaRR:



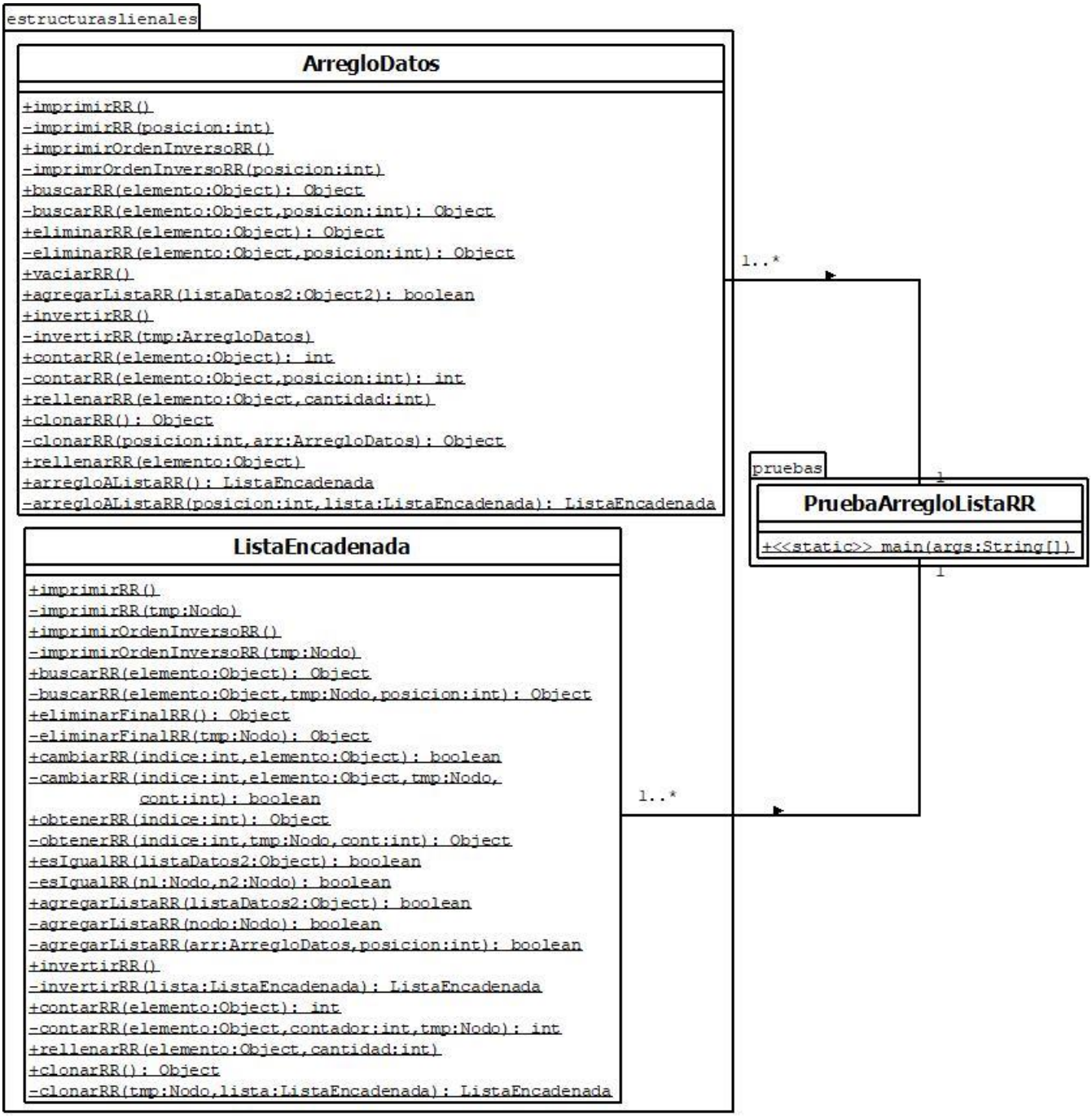




```
PRUEBA LISTARR:
Imprimir:
A -> B -> C -> D -> E -> null
Imprimir orden inverso:
E -> D -> C -> B -> A -> null
Buscar: 3
Eliminar final: E
A -> B -> C -> D -> null
Cambiar: true
A -> B -> X -> D -> null
Obtener: B
Clonar:
A -> B -> X -> D -> null
Son iguales: true
Agregar lista: true
A -> B -> X -> D -> B -> H -> null
Invertir:
H -> B -> D -> X -> B -> A -> null
Contar: 2
Rellenar:
H -> B -> D -> X -> B -> A -> L -> L -> L -> null
```

Código agregado:

La clase PruebaArregloListaRR es la única clase nueva, además todos los métodos subrayados y que se miran en el diagrama son nuevos.



Pre-evaluación:

Pre-Evaluación para prácticas de Laboratorio de Estructuras de Datos	PRE-EVALUACIÓN DEL ALUMNO
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.	Sí
INCLUYE LA DOCUMENTACIÓN GENERADA CON JAVADOC.	Sí
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).	100%
Observaciones:	

Conclusión:

Usar la recursividad para acceder a los datos del arreglo y las listas es más complicado porque se tienen que hacer 2 métodos para poder hacerlo de manera correcta, pero también estos métodos suelen ser más pequeños que los métodos que no usan la recursividad, por lo que en algunos casos pueden llegar a simplificar los métodos, aunque en otros pueda llegar a ser contraproducente.