Geeks México

BLOG DE PROGRAMACIÓN EN ESPAÑOL SOBRE JAVA, FRAMEWORKS, BASES DE DATOS, CÓMPUTO EN LA NUBE, ETC. EN ESPAÑOL Y EN INGLÉS.

Listas ligadas en Java paso a paso

y en Español

HACE 2 SEMANAS 3 COMENTARIOS

En este post explicaremos paso a paso como crear una lista ligada en Java, como sabemos existe un api llamado Collections api que se encuentra en el paquete java.util, nosotros no lo utilizaremos y crearemos una desde cero para entender como funciona.

Una lista ligada es un conjunto dinámico de elementos de cualquier tipo almacenados en memoria, estos pueden ser:

- Des ordenadas
- Ordenadas
- Con elementos únicos
- Con elementos duplicados
- Simplemente ligadas
- Doblemente ligadas

En este post explicaremos como crear una lista simplemente ligada.

Creando la clase Nodo

La clase nodo será la clase base y será utilizada para representar cada elemento en la lista.

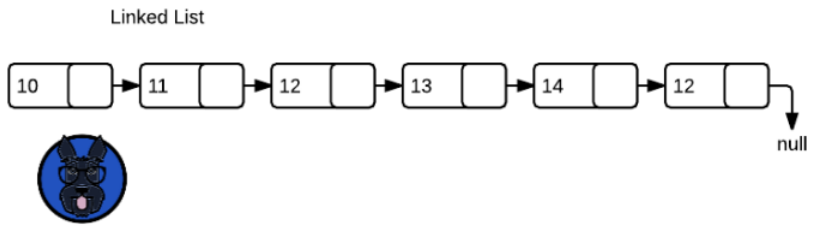
```
1  /**
2   * @author raidentrance
3   *
4   */
5  public class Node {
6      private Integer value;
7      private Node next;
8
9      public Node(Integer value) {
```

```
10          this.value = value;
11      }
12
13      public Integer getValue() {
14          return value;
15      }
16
17      public void setValue(Integer value) {
18          this.value = value;
19      }
20
21      public Node getNext() {
22          return next;
23      }
24
25      public void setNext(Node next) {
26          this.next = next;
27      }
28
29  }
```

Como se puede observar cada elemento de la lista contendrá dos atributos:

- *Integer value : Almacena el valor contenido en el elemento*
- *Node next : Almacena una referencia al nodo siguiente de la lista*

Las listas se llaman ligadas porque cada elemento contiene una referencia al nodo siguiente como se muestra en la siguiente imagen :



Creando la lista ligada

Una vez que definimos la estructura que tendrá el nodo el siguiente paso será crear una clase que represente las

acciones disponibles sobre esa lista, en este caso crearemos una clase llamada LinkedList.

```
import java.util.Optional;

/**
 * @author raidentrance
 *
 */
public class LinkedList {
    private Node head;

    public void append(Integer value) {
        if (head == null) {
            head = new Node(value);
            return;
        }
        Optional<Node> lastNode = getLastN
        Node node = lastNode.get();
        node.setNext(new Node(value));
    }

    public void print() {
        for (Node i = head; i != null; i =
            System.out.print(i.getValue()
        }
        System.out.println();
    }

    public Optional<Node> getLastNode() {
        if (head != null) {
            Node current = head;
            while (current.getNext() != nu
                current = current.getNext(
            }
            return Optional.of(current);
        } else {
            return Optional.empty();
        }
    }

    public void delete(Integer value) {
        System.out.printf("Deleting %d \n"
        if (head == null) {
            return;
        }
        if (head.getValue() == value) {
            head = head.getNext();
            return;
        }
        Node current = head;
        while (current != null && current.
            if (current.getNext().getValue
                current.setNext(current.ge
            }
            current = current.getNext();
        }

    }
}
```

Como se puede observar, las acciones soportadas en esta clase son las siguientes:

- *void append(Integer value) : Agrega un valor al final de la lista*
- *void print() : Imprime el contenido de la lista*
- *Optional getLastNode() : Devuelve una referencia al nodo final de la lista en caso de existir*
- *void delete(Integer value) : Borra el o los elementos que contengan el valor pasado como parámetro de la lista*

Probando la lista

Una vez que determinamos los nodos y las disponibles en la lista el siguiente paso será probarla para esto crearemos la siguiente clase:

```
/**
 * @author raidentrance
 *
 */
public class TestLinkedList {
    public static void main(String[] args)
        LinkedList list = new LinkedList()
        list.append(10);
        list.append(11);
        list.append(12);
        list.append(13);
        list.append(14);
        list.append(12);
        list.print();
        list.delete(14);
        list.print();
        list.delete(10);
        list.print();
        list.delete(12);
        list.print();
    }

}
```

Salida:

```
10  11  12  13  14  12
Deleting 14
10  11  12  13  12
Deleting 10
11  12  13  12
Deleting 12
11  13
```

En siguientes posts se explicará como utilizar una lista doblemente ligada así como otras estructuras de datos, Si te gusta el contenido compártelo y no olvides seguirnos en nuestras redes sociales https://twitter.com/geeks_mx (https://twitter.com/geeks_mx) y https://www.facebook.com/geeksJavaMexico/ (https://www.facebook.com/geeksJavaMexico/).

Autor: Alejandro Agapito Bautista

Twitter: @raidentrance

Contacto:raidentrance@gmail.com

3 Comentarios »

De que manera ayuda usar las clase java.util.Optional en los métodos de la lista?

★(https://geeks-mexico.com/2017/11/20/listas-ligadas-en-java-paso-a-paso-y-en-espanol/?like_comment=164&_wpnonce=3ac3d5756c)
Me gusta

Es útil para indicar que la respuesta es opcional, esto significa que es posible devolver una respuesta vacía. Es posible regresar null pero utilizar la clase Optional es una solución más legible, entendible y limpia, saludos.

★(https://geeks-mexico.com/2017/11/20/listas-ligadas-en-java-paso-a-paso-y-en-espanol/?like_comment=166&_wpnonce=27fe063c32)
Me gusta

Pingback: Listas doblemente ligadas en Java – Geeks México