LINKED LIST

By : Mahmoud Hussien

ARRAYS

- *An array is a collection of items of same data type stored at contiguous memory locations. ”Static Storage”*
- *Reserve a big chunk of the memory*
- *Indexed elements Starts from (0 to n-1)*
- *Good for accessing elements using index*
- *Bad in inserting and delete*

LINKED LIST

- A linked list is a data structure which can change during execution.
- Successive elements are connected by pointers.
- Last element points to **NULL**
- It can grow or shrink in size during execution of a program
- It can be made just as long as required
- It does not waste memory space.

REPRESENTATION OF LINKED LIST

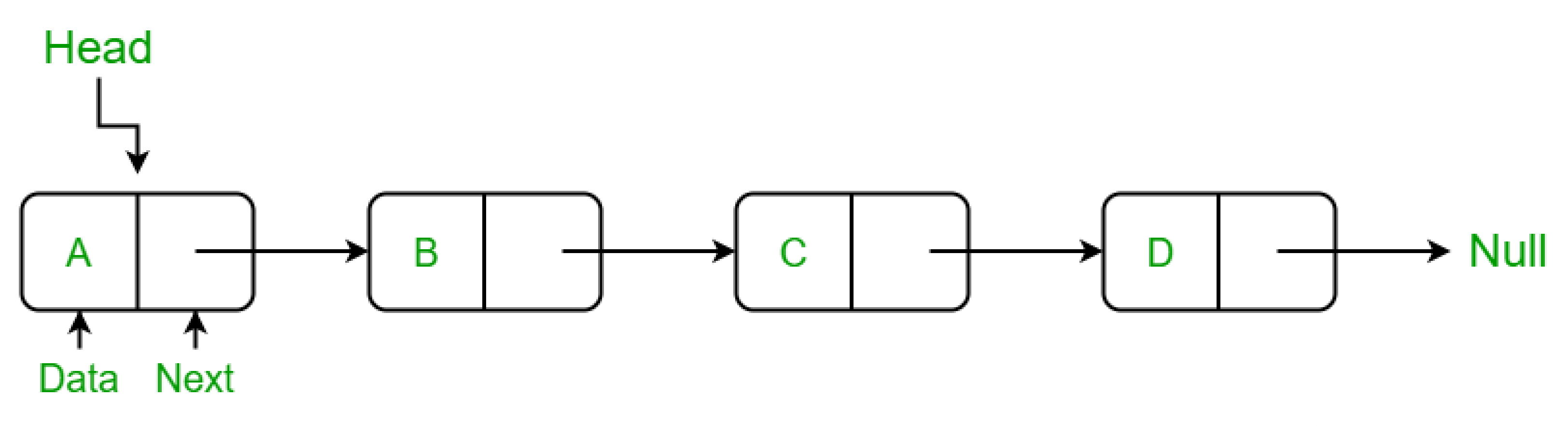
- A linked list is represented by a **pointer** to the first node of the linked list. The first node is called the **head** of the linked list. If the linked list is empty, then the value of the **head** points to **NULL**.
- Each node in a list consists of at least two parts:
- A **Data** Item (we can store integer, strings, or any type of data).
- **Pointer** (Or Reference) to the next node (connects one node to another) or An address of another node

TYPES OF LINKED LIST

- Linear Single Linked List
- Circular Linked List
- Double Linked List

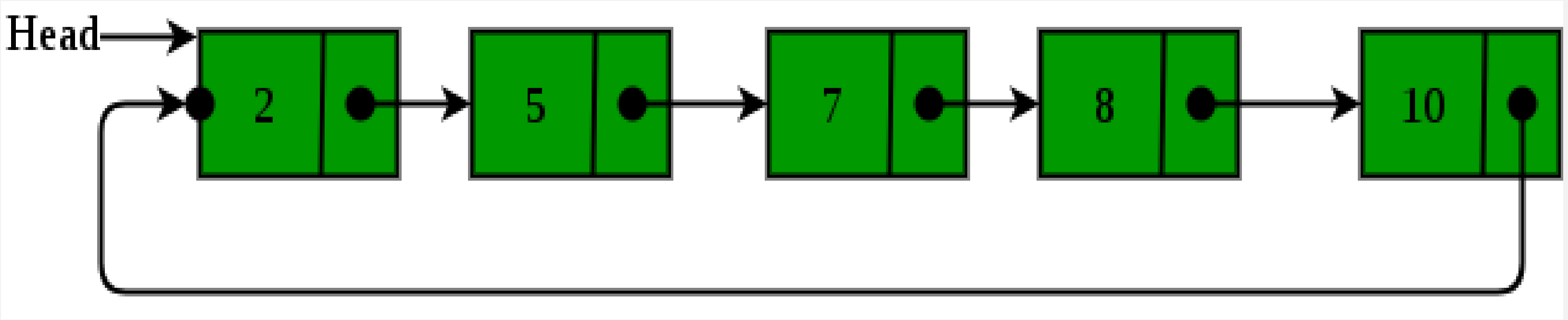
SINGLE LINKED LIST

- Linked List is a linear data structure. Unlike arrays, linked list elements are not stored at a contiguous location; the elements are linked using pointers. They include a series of connected nodes. Here, each node stores the data and the address of the next node.



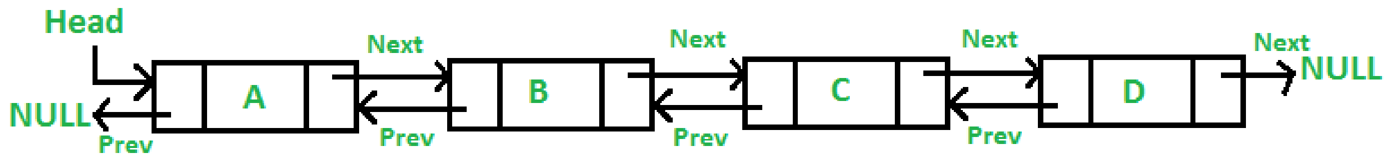
CIRCULAR LINKED LIST

- *The **circular linked list** is a linked list where all nodes are connected to form a circle. In a circular linked list, the first node and the last node are connected to each other which forms a circle. There is no NULL at the end.*



DOUBLE LINKED LIST

- A Doubly Linked List (**DLL**) contains an extra pointer, typically called the previous pointer, together with the next pointer and data which are there in the singly linked list.

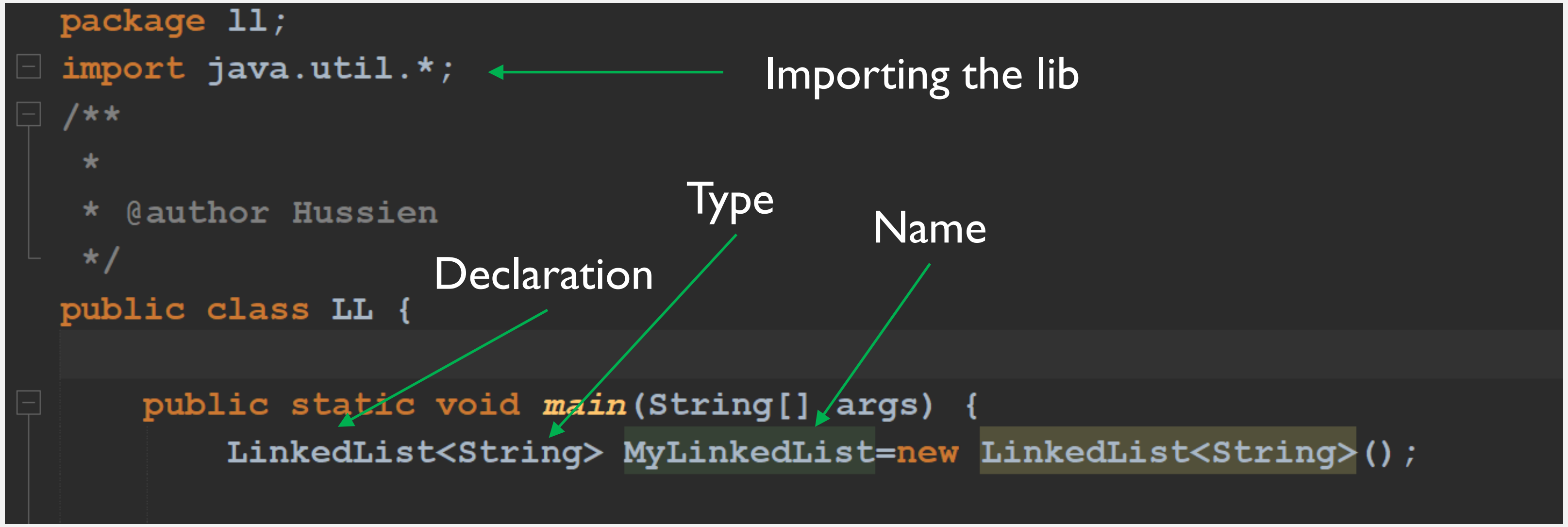


OPERATIONS

- **Deletion “<u>Remove</u>”**
- **Insertion “<u>Add</u>”**
- **Search “<u>Find</u>”**
- **Display “<u>Peek</u>”**

DECLARATION OF LINKED LIST USING BUILT IN LIBRARY

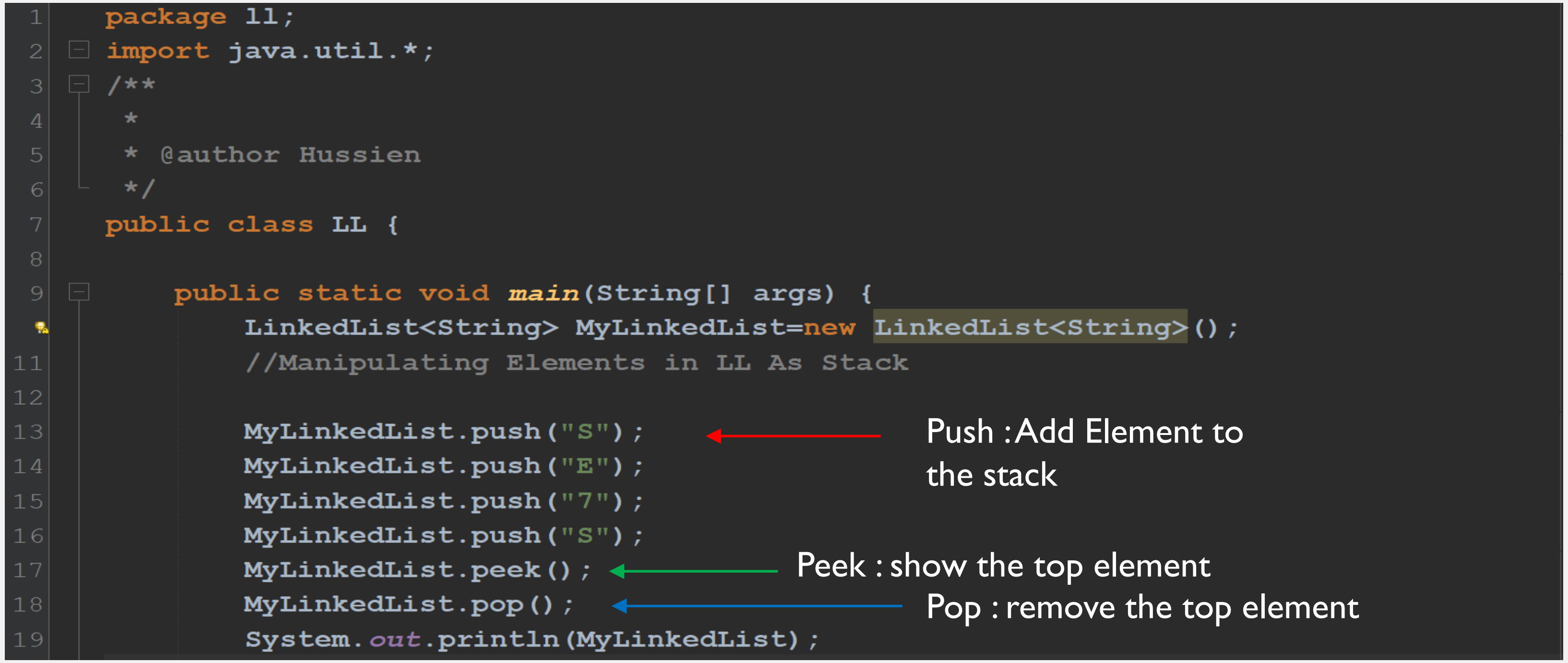
- First of all we need to import the Linked List Package in the util Lib
- Create a generic object from it in the main class
- Start manipulating



```
package ll;
import java.util.*;
/**
 *
 * @author Hussien
 */
public class LL {

    public static void main(String[] args) {
        LinkedList<String> MyLinkedList=new LinkedList<String>();
```

MANIPULATING ELEMENTS IN LL

- We can manipulate the linked list as "Stack/Queue/Linked-List"
- Manipulating elements as **Stack**



```
package ll;
import java.util.*;
/**
 *
 * @author Hussien
 */
public class LL {

    public static void main(String[] args) {
        LinkedList<String> MyLinkedList=new LinkedList<String>();
        //Manipulating Elements in LL As Stack

        MyLinkedList.push("S");
        MyLinkedList.push("E");
        MyLinkedList.push("7");
        MyLinkedList.push("S");
        MyLinkedList.peek();
        MyLinkedList.pop();
        System.out.println(MyLinkedList);
```

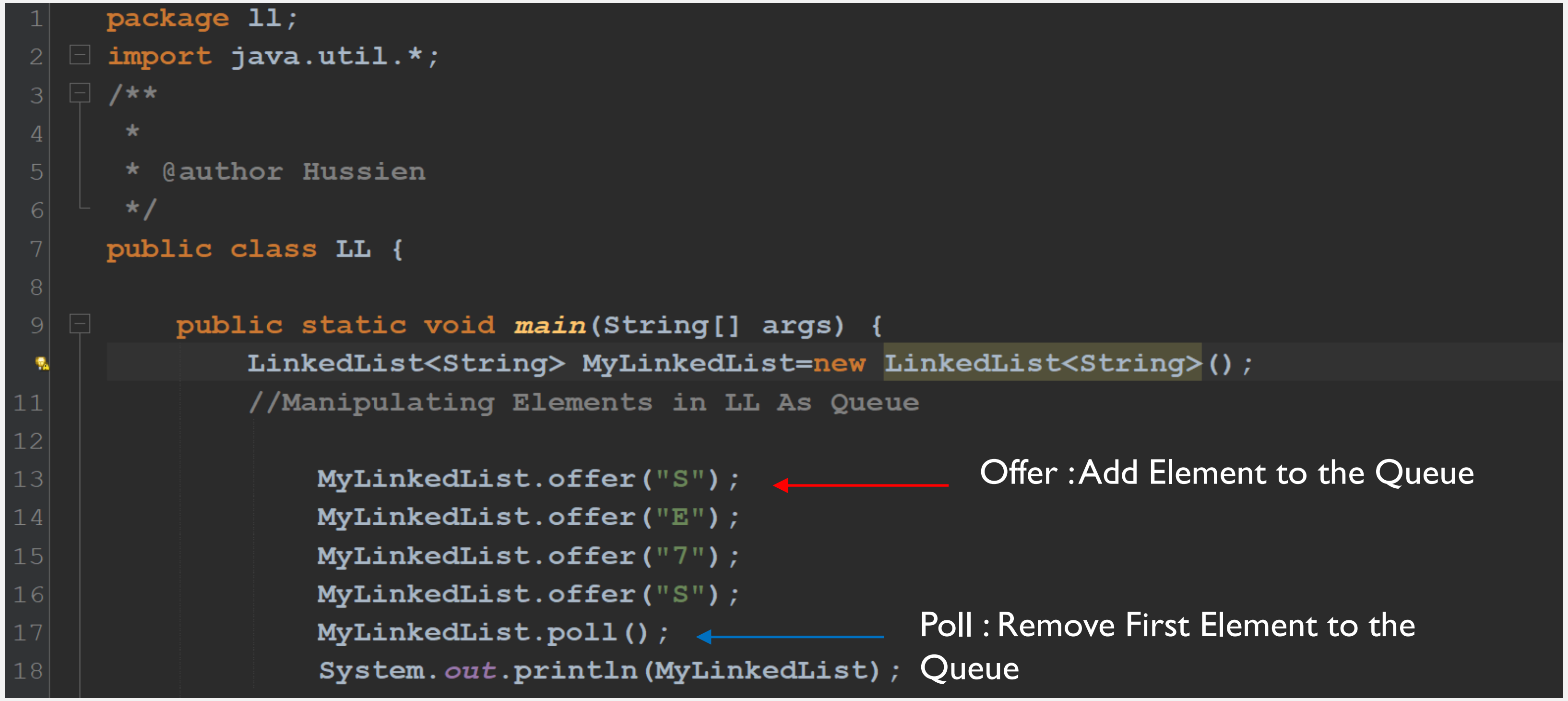
Push :Add Element to the stack

Peek : show the top element

Pop : remove the top element

MANIPULATING ELEMENTS IN LL

- We can manipulate the linked list as “Stack/Queue/Linked-List”
- Manipulating elements as **Queue**



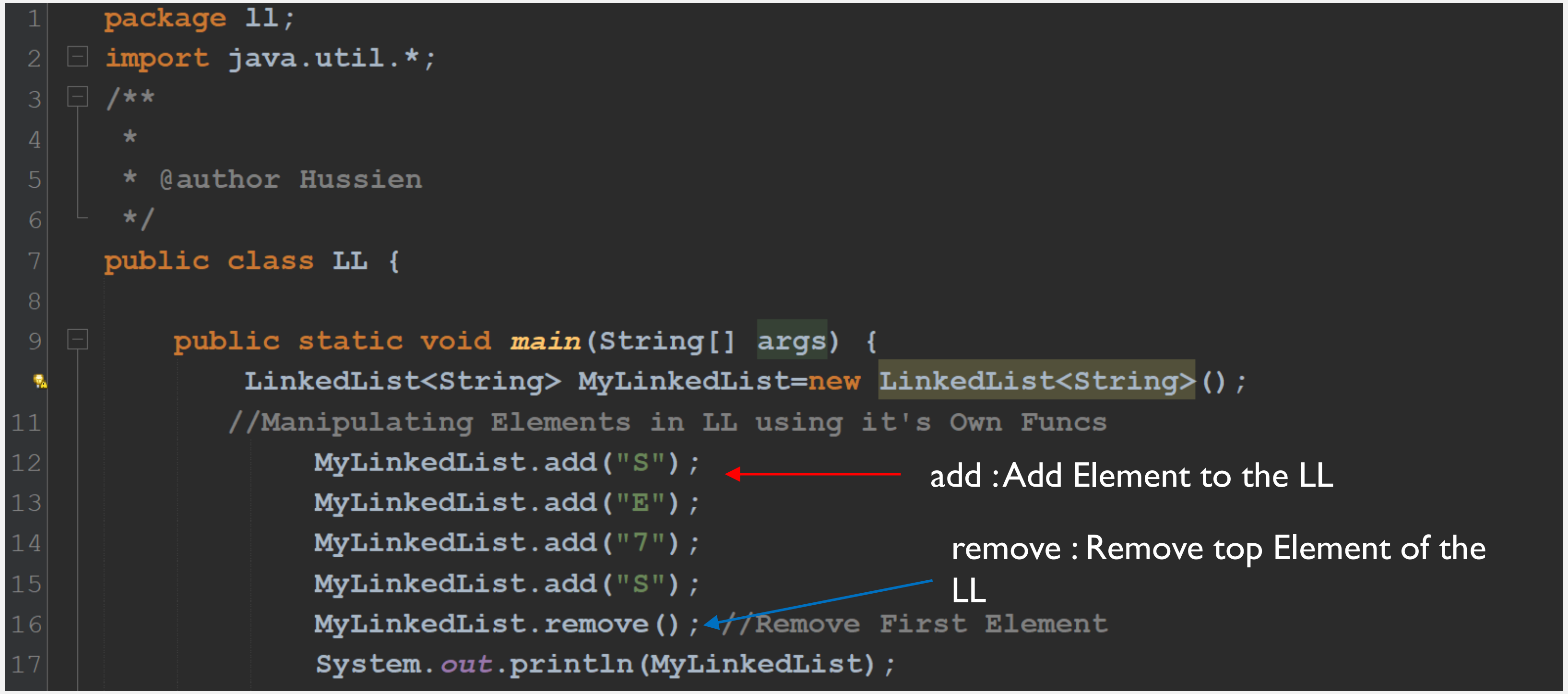
```java
package ll;
import java.util.*;
/**
 *
 * @author Hussien
 */
public class LL {

    public static void main(String[] args) {
        LinkedList<String> MyLinkedList=new LinkedList<String>();
        //Manipulating Elements in LL As Queue

            MyLinkedList.offer("S");
            MyLinkedList.offer("E");
            MyLinkedList.offer("7");
            MyLinkedList.offer("S");
            MyLinkedList.poll();
            System.out.println(MyLinkedList);
```

Offer :Add Element to the Queue

Poll : Remove First Element to the Queue

MANIPULATING ELEMENTS IN LL

- Using Basic Operations of Linked List to add/remove



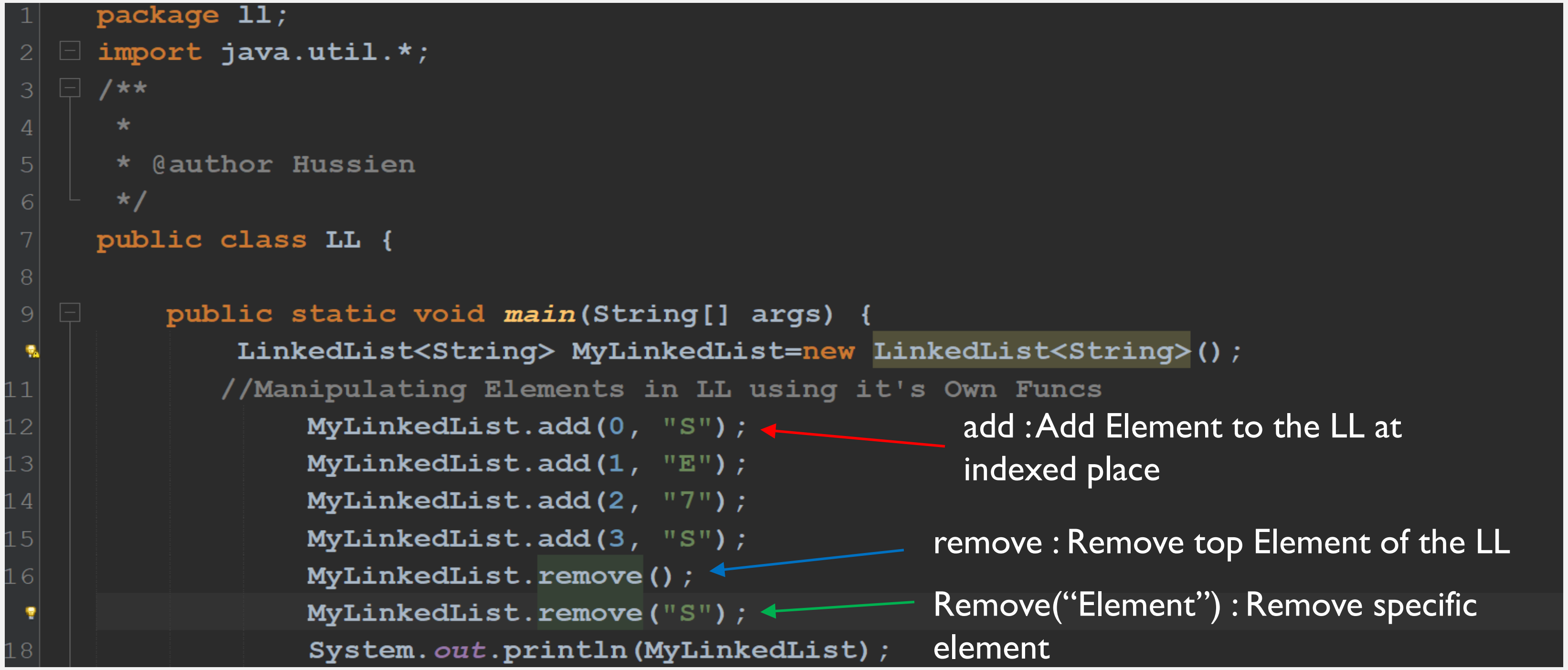
```java
package ll;
import java.util.*;
/**
 *
 * @author Hussien
 */
public class LL {

    public static void main(String[] args) {
        LinkedList<String> MyLinkedList=new LinkedList<String>();
       //Manipulating Elements in LL using it's Own Funcs
            MyLinkedList.add("S");
            MyLinkedList.add("E");
            MyLinkedList.add("7");
            MyLinkedList.add("S");
            MyLinkedList.remove();//Remove First Element
            System.out.println(MyLinkedList);
```

add : Add Element to the LL

remove : Remove top Element of the LL

MANIPULATING ELEMENTS IN LL

- Using Basic Operations of Linked List to add to specific location/remove specific element



```java
package ll;
import java.util.*;
/**
 *
 * @author Hussien
 */
public class LL {

    public static void main(String[] args) {
        LinkedList<String> MyLinkedList=new LinkedList<String>();
       //Manipulating Elements in LL using it's Own Funcs
            MyLinkedList.add(0, "S");
            MyLinkedList.add(1, "E");
            MyLinkedList.add(2, "7");
            MyLinkedList.add(3, "S");
            MyLinkedList.remove();
            MyLinkedList.remove("S");
            System.out.println(MyLinkedList);
```

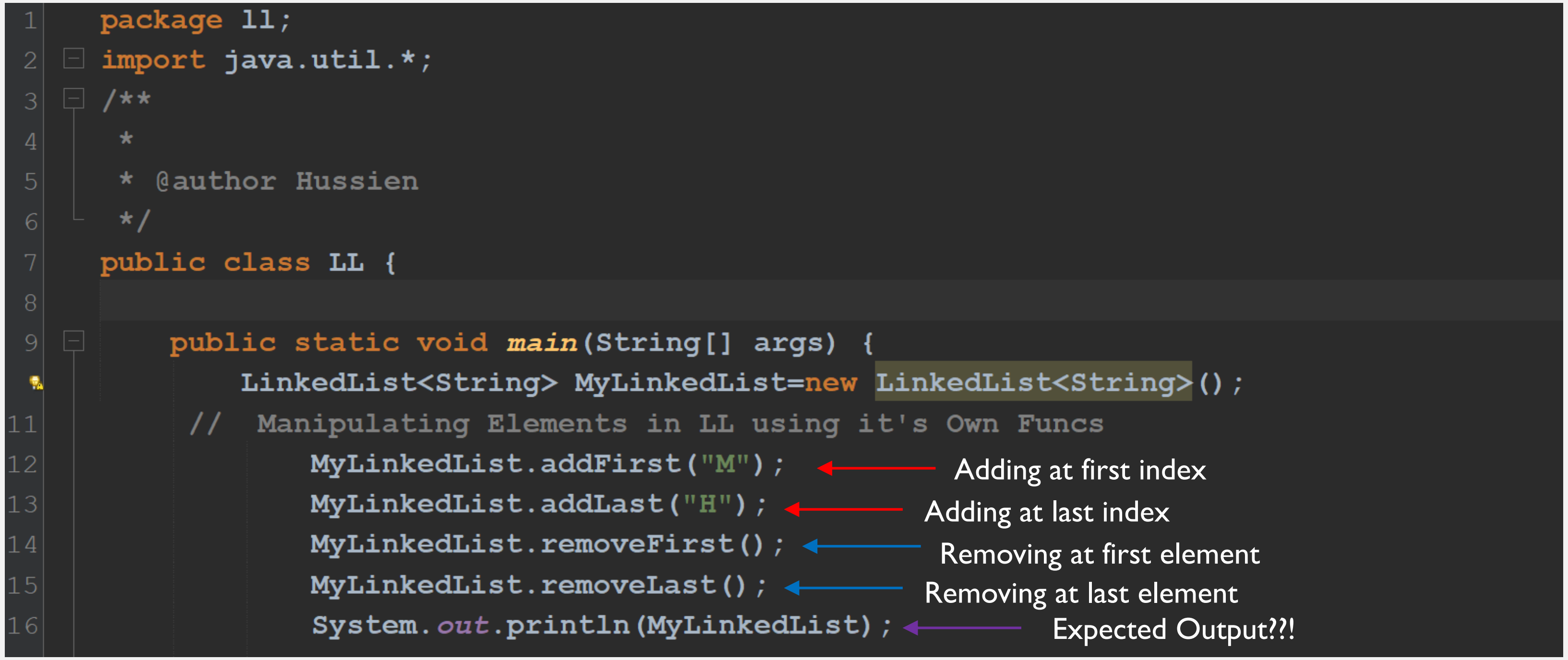
add :Add Element to the LL at indexed place

remove : Remove top Element of the LL

Remove(“Element”) : Remove specific element

MANIPULATING ELEMENTS IN LL

- Using Basic Operations of Linked List to add "First-Last"/remove "First-Last" element



```java
package ll;
import java.util.*;
/**
 *
 * @author Hussien
 */
public class LL {

    public static void main(String[] args) {
        LinkedList<String> MyLinkedList=new LinkedList<String>();
     //  Manipulating Elements in LL using it's Own Funcs
            MyLinkedList.addFirst("M");
            MyLinkedList.addLast("H");
            MyLinkedList.removeFirst();
            MyLinkedList.removeLast();
            System.out.println(MyLinkedList);
```

Adding at first index

Adding at last index

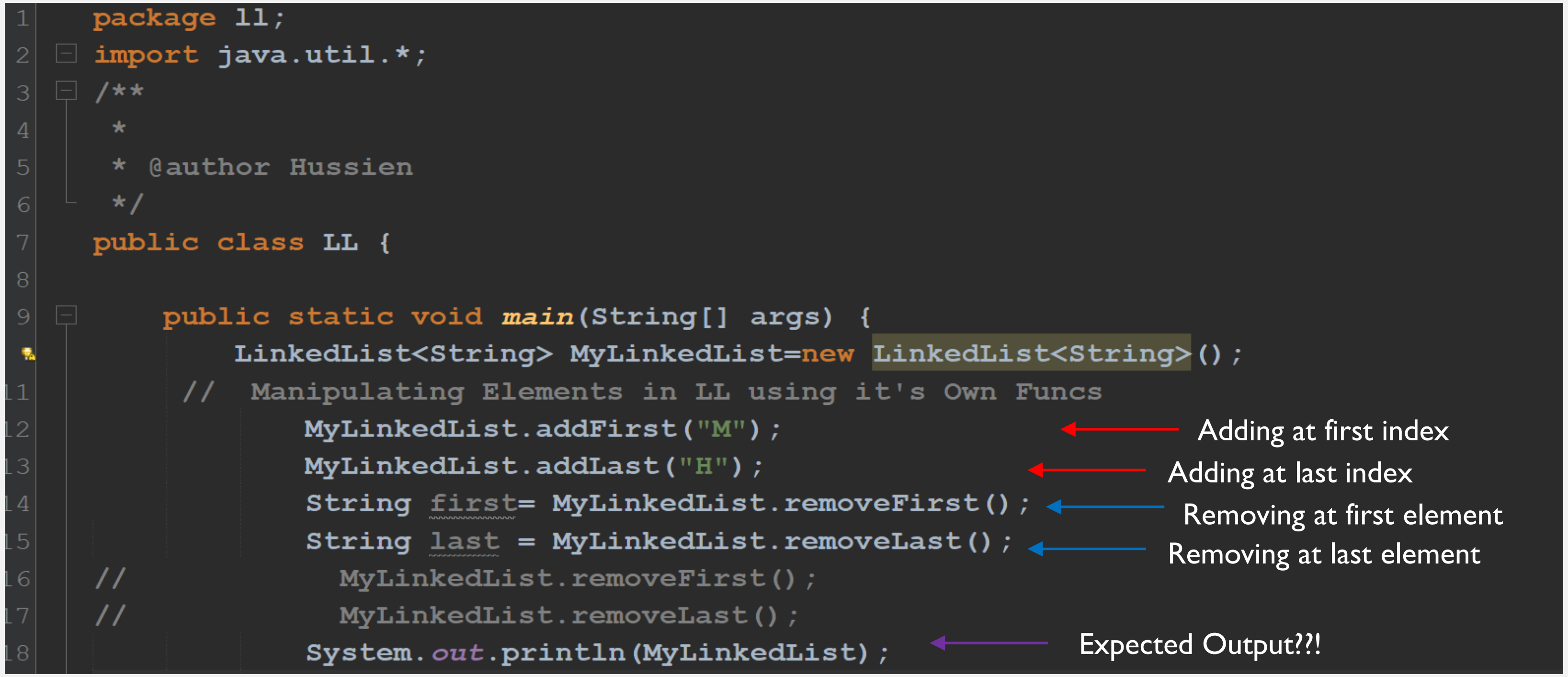
Removing at first element

Removing at last element

Expected Output??!

MANIPULATING ELEMENTS IN LL

- Using Basic Operations of Linked List to add “First-Last”/remove “First-Last” element



```java
package ll;
import java.util.*;
/**
 *
 * @author Hussien
 */
public class LL {

    public static void main(String[] args) {
        LinkedList<String> MyLinkedList=new LinkedList<String>();
      //  Manipulating Elements in LL using it's Own Funcs
            MyLinkedList.addFirst("M");
            MyLinkedList.addLast("H");
            String first= MyLinkedList.removeFirst();
            String last = MyLinkedList.removeLast();
//              MyLinkedList.removeFirst();
//              MyLinkedList.removeLast();
            System.out.println(MyLinkedList);
```

Adding at first index

Adding at last index

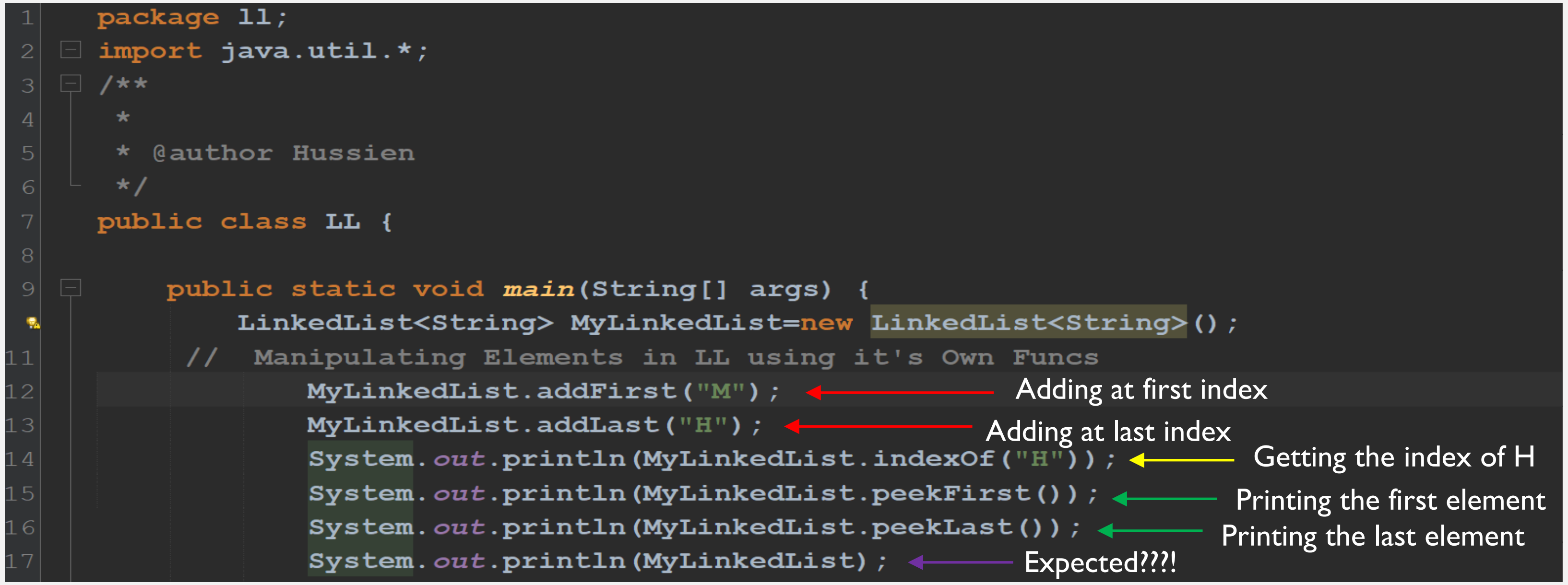
Removing at first element

Removing at last element

Expected Output??!

MANIPULATING ELEMENTS IN LL

- Using Basic Operations of Linked List to peek the first/last element and get index of specific element



```java
package ll;
import java.util.*;
/**
 *
 * @author Hussien
 */
public class LL {

    public static void main(String[] args) {
        LinkedList<String> MyLinkedList=new LinkedList<String>();
      //  Manipulating Elements in LL using it's Own Funcs
            MyLinkedList.addFirst("M");
            MyLinkedList.addLast("H");
            System.out.println(MyLinkedList.indexOf("H"));
            System.out.println(MyLinkedList.peekFirst());
            System.out.println(MyLinkedList.peekLast());
            System.out.println(MyLinkedList);
```

Adding at first index

Adding at last index

Getting the index of H

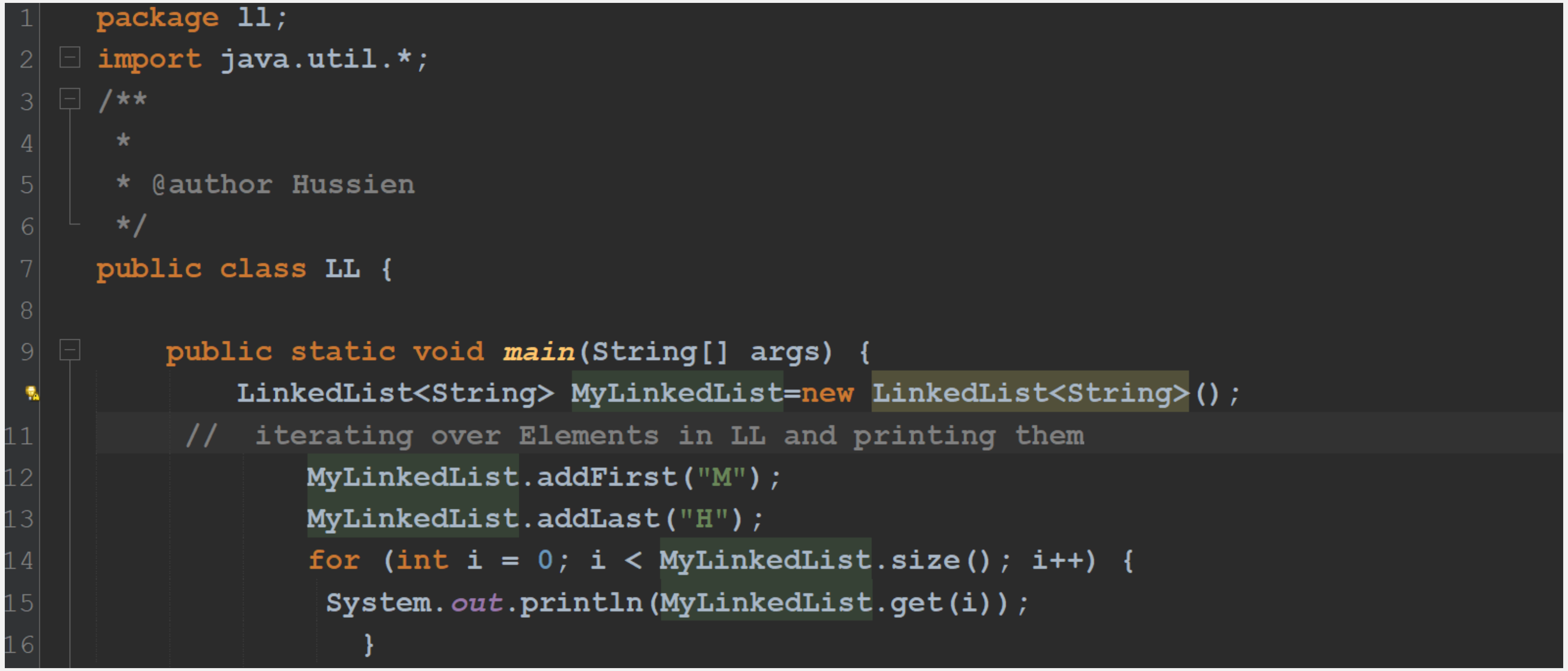
Printing the first element

Printing the last element

Expected???!

ITERATING OVER ELEMENTS

- Using a for loop to iterate over the elements of a Linked List



```
package ll;
import java.util.*;
/**
 *
 * @author Hussien
 */
public class LL {

    public static void main(String[] args) {
        LinkedList<String> MyLinkedList=new LinkedList<String>();
     //  iterating over Elements in LL and printing them
            MyLinkedList.addFirst("M");
            MyLinkedList.addLast("H");
            for (int i = 0; i < MyLinkedList.size(); i++) {
             System.out.println(MyLinkedList.get(i));
               }
```

APPLICATION OF LINKED LIST

1. **Image viewer** – Previous and next images are linked and can be accessed by the next and previous buttons.
2. **Previous and next page in a web browser** – We can access the previous and next URL searched in a web browser by pressing the back and next buttons since they are linked as a linked list.
3. **Music Player** – Songs in the music player are linked to the previous and next songs. So you can play songs either from starting or ending of the list.

PROS AND CONS OF LINKED LIST

- **Pros :**

- Dynamic Data Structure (allocates needed memory while running)
- Insertion and Deletion of Nodes is easy. O(1)
- No/Low memory waste

- **Cons :**

- Greater memory usage (additional pointer)
- No random access of elements (no index [i])
- Accessing/searching elements is more time consuming. O(n)

TASK #1

- Implement Linked List Class Using OOP with the following functions
- Insert
- Delete
- PrintList

# Reference to Look up : Implementation Of Linked List Using Java

TASK #2

- After Implementing Linked List Class do the following at the main function

- Count Number Of Elements
- Check if it contain a specific element
- Find the position of a specific element
- Get Minimum and Maximum of the list
- Get the Average of the elements of the list

CONTACT

- Facebook
- GitHub
- LinkedIn
- Gmail