|  |
| --- |
| Stacks |

**Stack**

A stack is a data structure which stores data in Last In First Out(LIFO) Method. Which means the data which is inserted at first is the one which will be removed at First.

40

20

90

65

70

25

80

80 is the element which goes in last and will be the first one when deleted.

40 is the element which goes in first and will be the last one when deleted.

**Implementation Of Stack**

Stack can be implemented in 2 different ways

Implementation Of Stack

Using Arrays

Using Linked List

**Implementation Of Stack Using Arrays**

*Stack.java*

|  |
| --- |
| public class Stack {      int arr[];      int topOfStack;  */\* Creation Of Stack \*/*      public Stack(int *size*) {          this.arr = new int[*size*];          topOfStack = -1;          System.out.println("Stack is Created With Size " + *size*);      }  */\* isEmpty() Method- Checks if stack is empty or not \*/*      public boolean isEmpty() {          if (topOfStack == -1) {              return true;          }          return false;      }  */\* isFull() Method - Checks if the stack is full or not \*/*      public boolean isFull() {          if (topOfStack == arr.length - 1) {              return true;          }          return false;      }  */\* push() method - inserts an element in the stack \*/*      public void push(int *element*) {          if (isFull()) {              System.out.println("Stack is Full");          } else {              arr[topOfStack + 1] = *element*;              System.out.println("Element inserted at index " + (topOfStack + 1));              topOfStack++;          }      }  */\* pop() method - removes the last inserted element from the stack. \*/*      public int pop() {          if (isEmpty()) {              System.out.println("Stack is empty!Nothing to pop");              return 0;          } else {              int topOfStackElement = arr[topOfStack];              System.out.println("Element popped is " + arr[topOfStack]);              topOfStack--;              return topOfStackElement;          }      }  */\* peek() method - returns the last inserted element \*/*      public int peek() {          if (isEmpty()) {              System.out.println("Stack is empty!Nothing to pop");              return 0;          } else {              int topOfStackElement = arr[topOfStack];              System.out.println(" Current Element is " + arr[topOfStack]);              return topOfStackElement;          }      }  */\* showStack() method - displays the entire stack \*/*      public void showStack() {          for (int i = 0; i <= topOfStack; i++) {              System.out.print(arr[i] + " ");          }          System.out.println();      }  */\* deleteStack() method - \*/*      public void deleteStack() {          topOfStack = -1;          arr = null;      }  } |

*main.java*

|  |
| --- |
| public class main {      public static void main(String[] *args*) {          Stack s1 = new Stack(5);          System.out.println(s1.isEmpty());          s1.push(5);          s1.push(2);          s1.push(7);          s1.push(8);          s1.push(9);          s1.showStack();          s1.pop();          s1.showStack();          System.out.println(s1.topOfStack);          s1.peek();          s1.pop();          s1.showStack();          System.out.println(s1.topOfStack);          s1.peek();      }  } |

*SinglyLinkedList.java*

|  |
| --- |
| public class SinglyLinkedList {      Node head;      Node tail;      int sizeOfLinkedList;  */\* Method 1- createLinkedList(int val) Creates A Linked List \*/*      public Node createLinkedList(int *value*) {          Node newNode = new Node();          newNode.value = *value*;          newNode.next = null;          head = newNode;          tail = newNode;          sizeOfLinkedList++;          System.out.println("Linked List  is created");          return head;      }  */\**  *\* Insertion Of Nodes In Linked list*  *\**  *\* 1.Insert Node At Beginning*  *\* 2.Insert Node At End*  *\* 3.Insert Node At Any Given Position*  *\*/*  */\* Method 2 - insertNodeAtFirst(int value) \*/*      public void insertNodeAtFirst(int *value*) {  */\* Case 1- If linked list is not created before \*/*          if (head == null) {              createLinkedList(*value*);              return;          }  */\* General Case \*/*          else {              Node newNode = new Node();              newNode.value = *value*;              newNode.next = head;              head = newNode;              sizeOfLinkedList++;              System.out.println("Node Inserted At First!");          }      }  */\* Method 3 - insertNodeAtLast(int value) \*/*      public void insertNodeAtLast(int *value*) {  */\* Case 1- If linked list is not created before \*/*          if (head == null) {              createLinkedList(*value*);              return;          }  */\* General Case \*/*          else {              Node newNode = new Node();              newNode.value = *value*;              tail.next = newNode;              newNode.next = null;              tail = newNode;              sizeOfLinkedList++;              System.out.println("Node Inserted At Last");          }      }  */\* Method 4 - insertNode(int value,int position) \*/*      public void insertNode(int *value*, int *position*) {  */\* Corner Cases \*/*  */\* Case 1- If linked list is not created before \*/*          if (head == null) {              createLinkedList(*value*);              return;          }  */\* Case 2- If position =1. Insertion At 1st position \*/*          if (*position* == 1) {              insertNodeAtFirst(*value*);              return;          }  */\* Case 3- If position > sizeOfLinkedList. Insertion At last position \*/*          if (*position* > sizeOfLinkedList) {              insertNodeAtLast(*value*);              return;          }  */\* Case 4- If position <=0 . Invalid Insertion Attempt \*/*          if (*position* <= 0) {              System.out.println("Invalid Position! Try Again. ");          }  */\* General Case \*/*          else {              Node tempNode = head;              for (int i = 1; i < *position* - 1; i++) {                  tempNode = tempNode.next;              }              Node toBeInserted = new Node();              toBeInserted.value = *value*;              Node nextNode = tempNode.next;              tempNode.next = toBeInserted;              toBeInserted.next = nextNode;              sizeOfLinkedList++;              System.out.println("Node inserted at position " + *position*);          }      }  */\* Method 5 - printList() \*/*      public void printList() {          Node tempNode = head;  */\**  *\* Using While Loop*  *\**  *\* while (tempNode != null) {*  *\* System.out.print(tempNode.value + " --> ");*  *\* tempNode = tempNode.next;*  *\* }*  *\* System.out.print("NULL \n");*  *\**  *\*/*  */\* Using For Loop \*/*          for (int i = 1; i <= sizeOfLinkedList; i++) {              System.out.print(tempNode.value + " --> ");              tempNode = tempNode.next;          }          System.out.print("NULL \n");      }  */\**  *\* Deletion Of Nodes In Linked list*  *\**  *\* 1.Delete Node From Beginning*  *\* 2.Delete Node From End*  *\* 3.Delete Node From Any Given Position*  *\*/*  */\* Deletion of Nodes From Beginning \*/*      public void deleteNodeFromFirst() {          if (head == null) {              System.out.println("Nothing To Delete! Linked List is Empty!");          } else if (sizeOfLinkedList == 1) {              head = null;              tail = null;              sizeOfLinkedList--;              System.out.println("Last Node Deleted!");          } else {              head = head.next;              sizeOfLinkedList--;              System.out.println("Node Deleted From First!");          }      }  */\* Deletion of Nodes From End \*/*      public void deleteNodeFromLast() {          if (head == null) {              System.out.println("Nothing To Delete! Linked List is Empty!");          } else if (sizeOfLinkedList == 1) {              head = null;              tail = null;              sizeOfLinkedList--;              System.out.println("Last Node Deleted!");          } else {              Node secondLastNode = head;              for (int i = 1; i < sizeOfLinkedList - 1; i++) {                  secondLastNode = secondLastNode.next;              }              tail = secondLastNode;              tail.next = null;              sizeOfLinkedList--;              System.out.println("Node Deleted From Last!");          }      }  */\* Deletion of Nodes From Any Given Position \*/*      public void deleteNode(int *position*) {  */\* Case 1 - If Linked List Doesnt Exist \*/*          if (head == null) {              System.out.println("Nothing To Delete! Linked List is Empty!");  */\* If position of Deletion is at position 1 \*/*          } else if (*position* == 1) {              deleteNodeFromFirst();          }  */\* If position of Deletion is greater than size of Linked List \*/*          else if (*position* >= sizeOfLinkedList) {              deleteNodeFromLast();          }  */\* If position is 0 or negative \*/*          else if (*position* <= 0) {              System.out.println("Invalid Index Provided! 0s and negatives not allowed");          }  */\* General Case \*/*          else {              Node tempNode = head;              for (int i = 1; i < *position* - 1; i++) {                  tempNode = tempNode.next;              }              Node toBeDeleted = tempNode.next;              Node nextNode = toBeDeleted.next;              tempNode.next = nextNode;              sizeOfLinkedList--;              System.out.println("Node Deleted From Position " + *position*);          }      }  */\**  *\* searchNode(int val)- Searching Of Nodes. Returns position of node if present.*  *\* Else returns -1*  *\*/*      public int searchNode(int *value*) {          Node tempNode = head;          for (int i = 1; i <= sizeOfLinkedList; i++) {              if (tempNode.value == *value*) {                  System.out.println("Value Found At Position " + i);                  return i;              }              tempNode = tempNode.next;          }          System.out.println("Value Not Found In Linked List! ");          return -1;      }  */\* Reverse Linked List - Reverses the linked list \*/*      public void reverseLinkedList() {          tail = head;          Node previousNode = null;          Node currentNode = head;          Node nextNode = head.next;          while (currentNode != null) {              currentNode.next = previousNode;              previousNode = currentNode;              currentNode = nextNode;              if (nextNode != null) {                  nextNode = nextNode.next;              }          }          System.out.println("LinkedList Reversed!");          head = previousNode;      }  */\* deleteLinkedList() - Deletes the entire linked list \*/*      public void deleteLinkedList() {          head = null;          tail = null;          sizeOfLinkedList = 0;          System.out.println("Linked List Deleted!");      }  } |

**Implementation Of Stack Using Linked List**

*Node.java*

|  |
| --- |
| public class Node {      int value;      Node next;  } |

*SinglyLinkedList.java*

|  |
| --- |
| public class SinglyLinkedList {      Node head;      Node tail;      int sizeOfLinkedList;  */\* Method 1- createLinkedList(int val) Creates A Linked List \*/*      public Node createLinkedList(int *value*) {          Node newNode = new Node();          newNode.value = *value*;          newNode.next = null;          head = newNode;          tail = newNode;          sizeOfLinkedList++;          System.out.println("Linked List  is created");          return head;      }  */\**  *\* Insertion Of Nodes In Linked list*  *\**  *\* 1.Insert Node At Beginning*  *\* 2.Insert Node At End*  *\* 3.Insert Node At Any Given Position*  *\*/*  */\* Method 2 - insertNodeAtFirst(int value) \*/*      public void insertNodeAtFirst(int *value*) {  */\* Case 1- If linked list is not created before \*/*          if (head == null) {              createLinkedList(*value*);              return;          }  */\* General Case \*/*          else {              Node newNode = new Node();              newNode.value = *value*;              newNode.next = head;              head = newNode;              sizeOfLinkedList++;              System.out.println("Node Inserted At First!");          }      }  */\* Method 3 - insertNodeAtLast(int value) \*/*      public void insertNodeAtLast(int *value*) {  */\* Case 1- If linked list is not created before \*/*          if (head == null) {              createLinkedList(*value*);              return;          }  */\* General Case \*/*          else {              Node newNode = new Node();              newNode.value = *value*;              tail.next = newNode;              newNode.next = null;              tail = newNode;              sizeOfLinkedList++;              System.out.println("Node Inserted At Last");          }      }  */\* Method 4 - insertNode(int value,int position) \*/*      public void insertNode(int *value*, int *position*) {  */\* Corner Cases \*/*  */\* Case 1- If linked list is not created before \*/*          if (head == null) {              createLinkedList(*value*);              return;          }  */\* Case 2- If position =1. Insertion At 1st position \*/*          if (*position* == 1) {              insertNodeAtFirst(*value*);              return;          }  */\* Case 3- If position > sizeOfLinkedList. Insertion At last position \*/*          if (*position* > sizeOfLinkedList) {              insertNodeAtLast(*value*);              return;          }  */\* Case 4- If position <=0 . Invalid Insertion Attempt \*/*          if (*position* <= 0) {              System.out.println("Invalid Position! Try Again. ");          }  */\* General Case \*/*          else {              Node tempNode = head;              for (int i = 1; i < *position* - 1; i++) {                  tempNode = tempNode.next;              }              Node toBeInserted = new Node();              toBeInserted.value = *value*;              Node nextNode = tempNode.next;              tempNode.next = toBeInserted;              toBeInserted.next = nextNode;              sizeOfLinkedList++;              System.out.println("Node inserted at position " + *position*);          }      }  */\* Method 5 - printList() \*/*      public void printList() {          Node tempNode = head;  */\**  *\* Using While Loop*  *\**  *\* while (tempNode != null) {*  *\* System.out.print(tempNode.value + " --> ");*  *\* tempNode = tempNode.next;*  *\* }*  *\* System.out.print("NULL \n");*  *\**  *\*/*  */\* Using For Loop \*/*          for (int i = 1; i <= sizeOfLinkedList; i++) {              System.out.print(tempNode.value + " --> ");              tempNode = tempNode.next;          }          System.out.print("NULL \n");      }  */\**  *\* Deletion Of Nodes In Linked list*  *\**  *\* 1.Delete Node From Beginning*  *\* 2.Delete Node From End*  *\* 3.Delete Node From Any Given Position*  *\*/*  */\* Deletion of Nodes From Beginning \*/*      public void deleteNodeFromFirst() {          if (head == null) {              System.out.println("Nothing To Delete! Linked List is Empty!");          } else if (sizeOfLinkedList == 1) {              head = null;              tail = null;              sizeOfLinkedList--;              System.out.println("Last Node Deleted!");          } else {              head = head.next;              sizeOfLinkedList--;              System.out.println("Node Deleted From First!");          }      }  */\* Deletion of Nodes From End \*/*      public void deleteNodeFromLast() {          if (head == null) {              System.out.println("Nothing To Delete! Linked List is Empty!");          } else if (sizeOfLinkedList == 1) {              head = null;              tail = null;              sizeOfLinkedList--;              System.out.println("Last Node Deleted!");          } else {              Node secondLastNode = head;              for (int i = 1; i < sizeOfLinkedList - 1; i++) {                  secondLastNode = secondLastNode.next;              }              tail = secondLastNode;              tail.next = null;              sizeOfLinkedList--;              System.out.println("Node Deleted From Last!");          }      }  */\* Deletion of Nodes From Any Given Position \*/*      public void deleteNode(int *position*) {  */\* Case 1 - If Linked List Doesnt Exist \*/*          if (head == null) {              System.out.println("Nothing To Delete! Linked List is Empty!");  */\* If position of Deletion is at position 1 \*/*          } else if (*position* == 1) {              deleteNodeFromFirst();          }  */\* If position of Deletion is greater than size of Linked List \*/*          else if (*position* >= sizeOfLinkedList) {              deleteNodeFromLast();          }  */\* If position is 0 or negative \*/*          else if (*position* <= 0) {              System.out.println("Invalid Index Provided! 0s and negatives not allowed");          }  */\* General Case \*/*          else {              Node tempNode = head;              for (int i = 1; i < *position* - 1; i++) {                  tempNode = tempNode.next;              }              Node toBeDeleted = tempNode.next;              Node nextNode = toBeDeleted.next;              tempNode.next = nextNode;              sizeOfLinkedList--;              System.out.println("Node Deleted From Position " + *position*);          }      }  */\**  *\* searchNode(int val)- Searching Of Nodes. Returns position of node if present.*  *\* Else returns -1*  *\*/*      public int searchNode(int *value*) {          Node tempNode = head;          for (int i = 1; i <= sizeOfLinkedList; i++) {              if (tempNode.value == *value*) {                  System.out.println("Value Found At Position " + i);                  return i;              }              tempNode = tempNode.next;          }          System.out.println("Value Not Found In Linked List! ");          return -1;      }  */\* Reverse Linked List - Reverses the linked list \*/*      public void reverseLinkedList() {          tail = head;          Node previousNode = null;          Node currentNode = head;          Node nextNode = head.next;          while (currentNode != null) {              currentNode.next = previousNode;              previousNode = currentNode;              currentNode = nextNode;              if (nextNode != null) {                  nextNode = nextNode.next;              }          }          System.out.println("LinkedList Reversed!");          head = previousNode;      }  */\* deleteLinkedList() - Deletes the entire linked list \*/*      public void deleteLinkedList() {          head = null;          tail = null;          sizeOfLinkedList = 0;          System.out.println("Linked List Deleted!");      }  } |

*Stack.java*

|  |
| --- |
| public class Stack {      SinglyLinkedList sll;      public Stack(int *value*) {          sll = new SinglyLinkedList();          sll.createLinkedList(*value*);          System.out.println("Stack Created!");      }      public boolean isEmpty() {          if (sll.head == null) {              return true;          }          else              return false;      }      public void push(int *value*) {          sll.insertNodeAtFirst(*value*);      }      public void pop() {          sll.deleteNodeFromFirst();      }      public void showStack() {          sll.printList();      }      public int peek() {          System.out.println(sll.head.value);          return sll.head.value;      }      public void deleteStack() {          sll.deleteLinkedList();      }  } |

*main.java*

|  |
| --- |
| public class main {      public static void main(String[] *args*) {          Stack s1 = new Stack(15);          s1.push(13);          s1.push(10);          s1.push(19);          s1.push(17);          s1.showStack();          s1.pop();          s1.showStack();          s1.peek();      }  } |