|  |
| --- |
| Linked Lists |

**Linked List**

Linked List is a data structure which is a sequential collection and it does not have to be in order. A linked list is made up of independent nodes that may contain any type of data. Each node has the reference to the next node in the link

Structural Representation Linked List

Head Node

Tail Node

Node

Node

* In Linked List, we cannot access any particular node directly except for the first node
* To access a node, we have to reach to the immediate preceding node by traversing the linked list all the way from the head

**Arrays vs Linked List**

|  |  |  |
| --- | --- | --- |
| **Sl** | **Array** | **Linked List** |
| 1 | Array is contiguous | Linked List in not contiguous. Each node of linked list is a separate object |
| 2 | Size of array needs to be predefined during creation. The size cannot be changed during program execution | Size of linked list is not needed to be predefined. We can add/remove as many nodes as required during the program execution |
| 3 | Accessing elements is very efficient in arrays. We can directly access any element directly | Accessing nodes in linked list is not direct except for the first node. We have to traverse the linked list all the way to the required node |

**Types Of Linked List**

Linked Lists

 Singly Linked List

Singly Circular Linked List

 Doubly Linked List

 Doubly Circular Linked List

**Singly Linked List**

 Head

xx001

 56

xx002

  75

 xx003

 89

 xx004

 52

null

xx004

Tail

xx001

xx002

xx003

xx004

xx000

* In Singly Linked List, we cannot can move forward to the next node, but cannot travel back to the previous node
* The tail points to the last node in case of singly linked list

**Circular Singly Linked List**

 Head

xx001

 56

xx002

  75

 xx003

 89

 xx004

 52

xx001

xx001

Tail

xx001

xx002

xx003

xx004

xx000

* The tail points to the head node in case of circular singly linked list completing a circle, hence is the name circular singly linked list

**Doubly Linked List**

 Head

xx001

 56

xx002

  75

 xx003

 89

 xx004

 52

null

xx004

Tail

xx001

xx002

xx003

xx004

xx000

* In Doubly Linked List, we can can move forward to the next node as well as come back to the previous node giving us extra flexibility
* The tail points to the last node in case of doubly linked list

**Doubly Circular Linked List**

 Head

xx001

 56

xx002

  75

 xx003

 89

 xx004

 52

null

xx000

Tail

xx001

xx002

xx003

xx004

xx000

* The tail points to the head node in case of circular doubly circular linked list completing a circle, hence is the name circular doubly linked list

|  |
| --- |
| Implementation Of Singly Linked Lists |

**Implementation Of Singly 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;      }  */\* deleteLinkedList() - Deletes the entire linked list \*/*      public void deleteLinkedList() {          head = null;          tail = null;          sizeOfLinkedList = 0;          System.out.println("Linked List Deleted!");      }  } |

*main.java*

|  |
| --- |
| public class main {      public static void main(String[] *args*) {  */\* Test 1 - Creating Linked list And Adding Nodes \*/*          SinglyLinkedList sll1 = new SinglyLinkedList();          sll1.insertNodeAtFirst(12);          sll1.insertNodeAtLast(15);          sll1.insertNodeAtLast(26);          sll1.insertNodeAtLast(34);          sll1.insertNodeAtLast(11);          sll1.insertNodeAtLast(53);          sll1.printList();  */\* Test 2 -Continuity Test \*/*          System.out.println(sll1.tail.value);          System.out.println(sll1.tail.next);  */\* Test 3 - Adding Nodes At Any Given Position \*/*          sll1.insertNode(45, 2);          sll1.printList();  */\* Test 4- Deletion Of Nodes \*/*          sll1.insertNodeAtLast(56);          sll1.insertNodeAtLast(13);          sll1.insertNodeAtLast(93);          sll1.printList();  */\* Test 4- Deletion Of Nodes From Last \*/*          sll1.deleteNodeFromLast();          System.out.println(sll1.tail.value);          sll1.printList();          sll1.deleteNodeFromLast();          System.out.println(sll1.tail.value);          sll1.printList();  */\* Continuity Test \*/*          sll1.insertNodeAtLast(54);          System.out.println(sll1.tail.value);          sll1.printList();          System.out.println(sll1.searchNode(54));      }  } |

**Time Complexity Analysis**

1.Creation Of Linked List

|  |
| --- |
| public Node createLinkedList(int *value*) {          Node newNode = new Node(); */\* O(1) \*/*          newNode.value = *value*; */\* O(1) \*/*          newNode.next = null; */\* O(1) \*/*          head = newNode; */\* O(1) \*/*          tail = newNode; */\* O(1) \*/*          sizeOfLinkedList++; */\* O(1) \*/*          System.out.println("Linked List  is created"); */\* O(1) \*/*          return head; */\* O(1) \*/*  */\*Total Time Complexity - O(1) \*/*      } |

2.Insertion Of Node In Singly Linked List

* Insertion At Beginning

|  |
| --- |
| public void insertNodeAtFirst(int *value*) {  */\* Case 1- If linked list is not created before \*/*          if (head == null) {              createLinkedList(*value*); */\* O(1) \*/*              return;          }  */\* General Case \*/*          else {              Node newNode = new Node();*/\* O(1) \*/*              newNode.value = *value*;*/\* O(1) \*/*              newNode.next = head;*/\* O(1) \*/*              head = newNode;*/\* O(1) \*/*              sizeOfLinkedList++;*/\* O(1) \*/*              System.out.println("Node Inserted At First!");*/\* O(1) \*/*          }  */\* Total Time Complexity - O(1) \*/*      } |

* Insertion At Given Position

|  |
| --- |
| public void insertNode(int *value*, int *position*) {  */\* Corner Cases \*/*  */\* Case 1- If linked list is not created before \*/*          if (head == null) {              createLinkedList(*value*); */\* O(1) \*/*              return;          }  */\* Case 2- If position =1. Insertion At 1st position \*/*          if (*position* == 1) {              insertNodeAtFirst(*value*);*/\* O(1) \*/*              return;          }  */\* Case 3- If position > sizeOfLinkedList. Insertion At last position \*/*          if (*position* > sizeOfLinkedList) {              insertNodeAtLast(*value*); */\* O(1) \*/*              return;          }  */\* Case 4- If position <=0 . Invalid Insertion Attempt \*/*          if (*position* <= 0) {              System.out.println("Invalid Position! Try Again. "); */\* O(1) \*/*          }  */\* General Case \*/*          else {              Node tempNode = head; */\* O(1) \*/*              for (int i = 1; i < *position* - 1; i++) {                  tempNode = tempNode.next; */\* O(n) \*/*              }              Node toBeInserted = new Node(); */\* O(1) \*/*              toBeInserted.value = *value*; */\* O(1) \*/*              Node nextNode = tempNode.next; */\* O(1) \*/*              tempNode.next = toBeInserted; */\* O(1) \*/*              toBeInserted.next = nextNode; */\* O(1) \*/*              sizeOfLinkedList++; */\* O(1) \*/*              System.out.println("Node inserted at position " + *position*); */\* O(1) \*/*          }  */\* Total Time Complexity - O(n) \*/*      } |

3.Deletion Of Node In Singly Linked List

* Deletion From Beginning

|  |
| --- |
| public void deleteNodeFromFirst() {          if (head == null) {              System.out.println("Nothing To Delete! Linked List is Empty!"); */\* O(1) \*/*          } else if (sizeOfLinkedList == 1) {              head = null;*/\* O(1) \*/*              tail = null;*/\* O(1) \*/*              sizeOfLinkedList--;*/\* O(1) \*/*              System.out.println("Last Node Deleted!");*/\* O(1) \*/*          } else {              head = head.next; */\* O(1) \*/*              sizeOfLinkedList--;*/\* O(1) \*/*              System.out.println("Node Deleted From First!");*/\* O(1) \*/*          }  */\* Total Time Complexity Is O(1) \*/*      } |

* Deletion From End Or Any Given Position

|  |
| --- |
| public void deleteNodeFromLast() {          if (head == null) {              System.out.println("Nothing To Delete! Linked List is Empty!"); */\* O(1) \*/*          } else if (sizeOfLinkedList == 1) {              head = null;*/\* O(1) \*/*              tail = null;*/\* O(1) \*/*              sizeOfLinkedList--;*/\* O(1) \*/*              System.out.println("Last Node Deleted!");*/\* O(1) \*/*          } else {              Node secondLastNode = head;              for (int i = 1; i < sizeOfLinkedList - 1; i++) {                  secondLastNode = secondLastNode.next; */\* O(n-1) \*/*              }              tail = secondLastNode;*/\* O(1) \*/*              tail.next = null;*/\* O(1) \*/*              sizeOfLinkedList--;*/\* O(1) \*/*              System.out.println("Node Deleted From Last!");*/\* O(1) \*/*          }  */\*Total Time Complexity Is O(n) \*/*      } |

4.Traversal Of Singly Linked List

|  |
| --- |
| 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++) { */\* O(n) \*/*              System.out.print(tempNode.value + " --> ");*/\* O(1) \*/*              tempNode = tempNode.next;*/\* O(1) \*/*          }          System.out.print("NULL \n");*/\* O(1) \*/*  */\* Total Time Complexity Is O(n) \*/*      } |

|  |
| --- |
| Implementation Of Singly Circular Linked Lists |

**Implementation Of Singly Circular Linked List**

*Node.java*

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

*SinglyCircularLinkedList.java*

|  |
| --- |
| public class SinglyCircularLinkedList {      Node head;      Node tail;      int sizeOfLinkedList;  */\* createLinkedList() - Creates a Linked List \*/*      public Node createLinkedList(String *value*) {          Node newNode = new Node();          newNode.value = *value*;          head = newNode;          tail = newNode;          tail.next = head;          sizeOfLinkedList++;          System.out.println("Linked List Successfully Created!");          return head;      }  */\* Insertion Of Nodes \*/*  */\**  *\* Insert Nodes At Beginnning*  *\* Insert Nodes At End*  *\* Insert Nodes At Any Given Position*  *\*/*  */\* insertNodesAtFirst() - Insert Nodes At Beginning \*/*      public void insertNodesAtFirst(String *value*) {  */\* If Linked List Is Not Created \*/*          if (head == null) {              createLinkedList(*value*);              return;          } else {              Node newNode = new Node();              newNode.value = *value*;              newNode.next = head;              head = newNode;              tail.next = head;              sizeOfLinkedList++;              System.out.println("Node inserted at First!");          }      }  */\* insertNodesAtLast(String value) - Insert Nodes At End \*/*      public void insertNodesAtLast(String *value*) {  */\* If Linked List Is Not Created \*/*          if (head == null) {              createLinkedList(*value*);              return;          } else {              Node newNode = new Node();              newNode.value = *value*;              tail.next = newNode;              tail = newNode;              tail.next = head;              sizeOfLinkedList++;              System.out.println("Node Inserted At Last");          }      }  */\* insertNode(String value,int position ) \*/*      public void insertNode(String *value*, int *position*) {  */\* If Linked List Is Not Created \*/*          if (head == null) {              createLinkedList(*value*);              return;          }  */\* If position is 1.Insert at first \*/*          else if (*position* == 1) {              insertNodesAtFirst(*value*);          }  */\* If position is greater than size of linked list, insert at last \*/*          else if (*position* > sizeOfLinkedList) {              insertNodesAtLast(*value*);          }  */\* If position is negative or 0, invalid insertion \*/*          else if (*position* <= 0) {              System.out.println("Invalid Position. 0 and negatives are not allowed");          }  */\* General Case \*/*          else {              Node tempNode = head;              for (int i = 1; i < *position* - 1; i++) {                  tempNode = tempNode.next;              }              Node nextNode = tempNode.next;              Node toBeInserted = new Node();              toBeInserted.value = *value*;              toBeInserted.next = nextNode;              tempNode.next = toBeInserted;              sizeOfLinkedList++;              System.out.println("Node Added At Position " + *position*);          }      }  */\* printList() -Prints the linked list \*/*      public void printList() {          Node tempNode = head;          for (int i = 1; i <= sizeOfLinkedList; i++) {              System.out.print(tempNode.value + " --> ");              tempNode = tempNode.next;          }          System.out.print("HEAD \n");      }  */\* Delete Nodes \*/*  */\* deleteNodeFromFirst() - Deletes Node From Beginning Of THe List \*/*      public void deleteNodeFromFirst() {  */\* If Linked List Is Not Created \*/*          if (head == null) {              System.out.println("Linked List Doesnt Exist! Nothing To Delete");          }  */\* If Linked List has only one node \*/*          else if (sizeOfLinkedList == 1) {              head = null;              tail = null;              sizeOfLinkedList--;              System.out.println("Last Node Deleted! Nothing To Delete!");          }  */\* General Case \*/*          else {              head = head.next;              tail.next = head;              sizeOfLinkedList--;              System.out.println("Node deleted From First!");          }      }  */\* deleteNodeFromLast() - Deletes Node From End Of THe List \*/*      public void deleteNodeFromLast() {  */\* If Linked List Is Not Created \*/*          if (head == null) {              System.out.println("Linked List Doesnt Exist! Nothing To Delete");          }  */\* If Linked List has only one node \*/*          else if (sizeOfLinkedList == 1) {              head = null;              tail = null;              sizeOfLinkedList--;              System.out.println("Last Node Deleted! Nothing To Delete!");          }          else {              Node secondLastNode = head;              for (int i = 1; i < sizeOfLinkedList - 1; i++) {                  secondLastNode = secondLastNode.next;              }              secondLastNode.next = head;              tail = secondLastNode;              sizeOfLinkedList--;              System.out.println("Node Deleted From Last!");          }      }      public void deleteNode(int *position*) {  */\* If Linked List Is Not Created \*/*          if (head == null) {              System.out.println("Linked List Doesnt Exist! Nothing To Delete");          }  */\* If Linked List has only one node \*/*          else if (sizeOfLinkedList == 1) {              head = null;              tail = null;              sizeOfLinkedList--;              System.out.println("Last Node Deleted! Nothing To Delete!");          }  */\* If position = 1, Delete From First \*/*          else if (*position* == 1) {              deleteNodeFromFirst();          }  */\* If position >=sizeOfLinkedList, Delete From Last \*/*          else if (*position* >= sizeOfLinkedList) {              deleteNodeFromLast();          }  */\* If position ,=0, invalid operation \*/*          else if (*position* <= 0) {              System.out.println("Invalid Option! 0s and negatives not accepted!");          } else {              Node tempNode = head;              for (int i = 1; i < *position* - 1; i++) {                  tempNode = tempNode.next;              }              Node toBeDeleted = tempNode.next;              Node nextNode = tempNode.next.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(String *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;      }  */\* deleteLinkedList() - Deletes the entire linked list \*/*      public void deleteLinkedList() {          head = null;          tail = null;          sizeOfLinkedList = 0;          System.out.println("Linked List Deleted!");      }  } |

*main.java*

|  |
| --- |
| public class main {      public static void main(String[] *args*) {          SinglyCircularLinkedList scll1 = new SinglyCircularLinkedList();          scll1.createLinkedList("Susi");          scll1.insertNodesAtLast("Fedora");          scll1.insertNodesAtLast("Red Hat Linux");          scll1.insertNodesAtFirst("Ubuntu");          scll1.insertNodesAtFirst("Parrot OS");          scll1.printList();          scll1.insertNode("Kali Linux", 4);          scll1.printList();  */\* Continuity Test \*/*          System.out.println(scll1.tail.value);          System.out.println(scll1.tail.next.value);  */\* Deletion Test \*/*          scll1.deleteNode(4);          scll1.printList();  */\* Continuity Test \*/*          System.out.println(scll1.tail.value);          System.out.println(scll1.tail.next.value);          System.out.println(scll1.searchNode("Red Hat Linux"));      }  } |

|  |
| --- |
| Implementation Of Doubly Linked Lists |

**Implementation Of Doubly Linked List**

*Node.java*

|  |
| --- |
| public class Node {      String value;      Node next;      Node prev;  } |

*DoublyLinkedList.java*

|  |
| --- |
| public class DoublyLinkedList {      Node head;      Node tail;      int sizeOfLinkedList;      public Node createLinkedList(String *value*) {          Node newNode = new Node();          newNode.value = *value*;          head = newNode;          tail = newNode;          tail.next = null;          sizeOfLinkedList++;          System.out.println("Linked List Created!");          return head;      }  */\* getSizeOfLinkedList() - Returns the size of the linked list \*/*      public int getSizeOfLinkedList() {          return sizeOfLinkedList;      }  */\**  *\* Inserting Nodes*  *\**  *\* 1.insertNodeAtFirst()*  *\* 2.insertNodeAtLast()*  *\* 3.insertNode()*  *\*/*  */\* insertNodeAtFirst(String value) \*/*      public void insertNodeAtFirst(String *value*) {  */\* Case 1- If linked list doesnt exist \*/*          if (head == null) {              createLinkedList(*value*);              return;          }  */\* General Case \*/*          else {              Node newNode = new Node();              newNode.value = *value*;              newNode.next = head;              head.prev = newNode;              head = newNode;              sizeOfLinkedList++;              System.out.println("New node added at First!");          }      }  */\* insertNodeAtLast(String value) \*/*      public void insertNodeAtLast(String *value*) {  */\* Case 1- If linked list doesnt exist \*/*          if (head == null) {              createLinkedList(*value*);              return;          }  */\* General Case \*/*          else {              Node newNode = new Node();              newNode.value = *value*;              tail.next = newNode;              newNode.prev = tail;              newNode.next = null;              tail = newNode;              sizeOfLinkedList++;              System.out.println("Node Added At Last!");          }      }  */\* insertNode(String value,int position) \*/*      public void insertNode(String *value*, int *position*) {  */\* Case 1- If linked list doesnt exist \*/*          if (head == null) {              createLinkedList(*value*);              return;          }  */\* If position =1 , i.e insertion at first position \*/*          else if (*position* == 1) {              insertNodeAtFirst(*value*);          }  */\* If position >=sizeOfLinkedList , i.e insertion at last position \*/*          else if (*position* >= sizeOfLinkedList) {              insertNodeAtLast(*value*);          } else if (*position* <= 0) {              System.out.println("Negative Values And 0 not accepted!");          }  */\* General Case \*/*          else {              Node newNode = new Node();              newNode.value = *value*;              Node tempNode = head;              for (int i = 1; i < *position* - 1; i++) {                  tempNode = tempNode.next;              }              Node nextNode = tempNode.next;              tempNode.next = newNode;              newNode.prev = tempNode;              newNode.next = nextNode;              nextNode.prev = newNode;              sizeOfLinkedList++;              System.out.println("Node inserted at position " + *position*);          }      }  */\* printList() - prints the entire linked list \*/*      public void printList() {          Node tempNode = head;          for (int i = 1; i <= sizeOfLinkedList; i++) {              if (i == sizeOfLinkedList) {                  System.out.print(tempNode.value + " --> ");                  continue;              }              System.out.print(tempNode.value + " <--> ");              tempNode = tempNode.next;          }          System.out.print("NULL \n");      }  */\**  *\* Deleting Nodes*  *\**  *\* 1.deleteNodeFromFirst()*  *\* 2.deleteNodeFromLast()*  *\* 3.deleteNode()*  *\*/*  */\**  *\* deleteNodeFromFirst() - Deletes a node from the beginning of the linked list*  *\*/*      public void deleteNodeFromFirst() {  */\* If Linked List Doesnt Exist \*/*          if (head == null) {              System.out.println("Nothing To Delete!");          }  */\* If only one node exists \*/*          else if (sizeOfLinkedList == 1) {              head = null;              tail = null;              sizeOfLinkedList--;              System.out.println("Last Node Deleted! Nothing Else To Delete!");          }  */\* General Case \*/*          else {              Node firstNode = head;              Node newFirstNode = head.next;              firstNode.next = null;              newFirstNode.prev = null;              head = newFirstNode;              sizeOfLinkedList--;              System.out.println("Deleted Node From First");          }      }  */\**  *\* deleteNodeFromLast() - Deletes a node from the end of the linked list*  *\*/*      public void deleteNodeFromLast() {  */\* If Linked List Doesnt Exist \*/*          if (head == null) {              System.out.println("Nothing To Delete!");          }  */\* If only one node exists \*/*          else if (sizeOfLinkedList == 1) {              head = null;              tail = null;              sizeOfLinkedList--;              System.out.println("Last Node Deleted! Nothing Else To Delete!");          }  */\* General Case \*/*          else {              Node secondLastNode = tail.prev;              secondLastNode.next = null;              tail.prev = null;              tail = secondLastNode;              sizeOfLinkedList--;              System.out.println("Last Node Deleted!");          }      }      public void deleteNode(int *position*) {  */\* If Linked List Doesnt Exist \*/*          if (head == null) {              System.out.println("Nothing To Delete!");          }  */\* If only one node exists \*/*          else if (sizeOfLinkedList == 1) {              head = null;              tail = null;              sizeOfLinkedList--;              System.out.println("Last Node Deleted! Nothing Else To Delete!");          }  */\* If position =1. Delete node from first \*/*          else if (*position* == 1) {              deleteNodeFromFirst();          }  */\* If position >=size of linked list. Delete node from last \*/*          else if (*position* >= sizeOfLinkedList) {              deleteNodeFromLast();          }  */\* If position <=0 . Invalid case \*/*          else if (*position* <= 0) {              System.out.println("Invalid position supplied");          }          else {              Node tempNode = head;              for (int i = 1; i < *position* - 1; i++) {                  tempNode = tempNode.next;              }              Node nextNode = tempNode.next.next;              Node toBeDeleted = tempNode.next;              toBeDeleted.next = null;              toBeDeleted.prev = null;              tempNode.next = nextNode;              nextNode.prev = tempNode;              sizeOfLinkedList--;              System.out.println("Node deleted from position " + *position*);          }      }  */\* searchNode(String value) - searches a Node from the list \*/*      public int searchNode(String *value*) {          Node tempNode = head;          for (int i = 1; i <= sizeOfLinkedList; i++) {              if (tempNode.value == *value*) {                  System.out.println("Node found at position " + i);                  return i;              }          }          System.out.println("Node doesnt exist in this Linked List!");          return -1;      }  */\* deleteLinkedList() -Deletes the linked list \*/*      public void deleteLinkedList() {          head = null;          tail = null;          sizeOfLinkedList = 0;          System.out.println("Linked List Successfully Deleted!");      }  } |

*main.java*

|  |
| --- |
| public class main {      public static void main(String[] *args*) {  */\* Creation of linked list and insertion at first/last \*/*          DoublyLinkedList dll1 = new DoublyLinkedList();          dll1.createLinkedList("Cypher");          dll1.insertNodeAtFirst("Sage");          dll1.insertNodeAtLast("Killjoy");          dll1.insertNodeAtLast("Deadlock");          dll1.printList();  */\* Insertion at a particular position \*/*          dll1.insertNode("Chamber", 4);          dll1.printList();  */\* Continuity Test \*/*          System.out.println(dll1.tail.value);          System.out.println(dll1.head.value);          DoublyLinkedList dll2 = new DoublyLinkedList();          dll2.createLinkedList("Sova");          dll2.insertNodeAtFirst("Breach");          dll2.insertNodeAtLast("Skye");          dll2.insertNodeAtLast("Fade");          dll2.insertNodeAtLast("Gekko");          dll2.printList();  */\* Deletion From First, Last, \*/*          dll2.deleteNodeFromLast();          dll2.printList();  */\* Continuity Test \*/*          System.out.println(dll2.tail.value);          System.out.println(dll2.head.value);          dll2.deleteNodeFromFirst();          dll2.printList();  */\* Continuity Test \*/*          System.out.println(dll2.tail.value);          System.out.println(dll2.head.value);          dll1.printList();          dll1.deleteNode(3);          dll1.printList();  */\* Continuity Test \*/*          System.out.println(dll2.tail.value);          System.out.println(dll2.head.value);      }  } |

|  |
| --- |
| Implementation Of Doubly Linked Lists |

**Implementation Of Doubly Linked List**

*Node.java*

|  |
| --- |
| public class Node {      String value;      Node next;      Node prev;  } |

*DoublyCircularLinkedList.java*

|  |
| --- |
| public class DoublyCircularLinkedList {      Node head;      Node tail;      int sizeOfLinkedList;  */\* getSizeOfLinkedList() \*/*  */\* createLinkedList() creates a Doubly Linked List \*/*      public void createLinkedList(String *value*) {          Node newNode = new Node();          newNode.value = *value*;          head = newNode;          tail = newNode;          tail.next = head;          sizeOfLinkedList++;          System.out.println("Linked List is Created!");      }      public int getSizeOfLinkedList() {          return sizeOfLinkedList;      }  */\**  *\* Insert Node*  *\**  *\* 1.Insert Node At First*  *\* 2.Insert Node At Last*  *\* 3.Insert Node At Any Given Position*  *\*/*      public void insertNodeAtFirst(String *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.prev = newNode;              head = newNode;              tail.next = head;              sizeOfLinkedList++;              System.out.println("Node inserted At First!");          }      }  */\* Insert Node At Last! \*/*      public void insertNodeAtLast(String *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.prev = tail;              tail = newNode;              tail.next = head;              sizeOfLinkedList++;              System.out.println("Node inserted At Last!");          }      }  */\* insertNode(String value,int position) Insert Node At Any Given Position \*/*      public void insertNode(String *value*, int *position*) {  */\* Case 1- If linked list is not created before \*/*          if (head == null) {              createLinkedList(*value*);              return;          }  */\* Case 2- If node is added at the beginning of the linked list \*/*          else if (*position* == 1) {              insertNodeAtFirst(*value*);          }  */\* Case 3- If node is added at the end of the linked list \*/*          else if (*position* > sizeOfLinkedList) {              insertNodeAtLast(*value*);          }  */\* Case 4- If position is negative or zero. Invalid Case \*/*          else {              insertNodeAtLast(*value*);          }      }  */\* printList() - Prints the linked list \*/*      public void printList() {          Node tempNode = head;          int count = 1;          while (count != sizeOfLinkedList + 1) {              if (count == sizeOfLinkedList) {                  System.out.print(tempNode.value + " --> ");                  tempNode = tempNode.next;                  count++;                  continue;              }              System.out.print(tempNode.value + " <--> ");              tempNode = tempNode.next;              count++;          }          System.out.print("HEAD \n");  */\**  *\* for (int i = 1; i <= sizeOfLinkedList; i++) {*  *\**  *\* if (i == sizeOfLinkedList) {*  *\* System.out.print(tempNode.value + " --> ");*  *\* tempNode = tempNode.next;*  *\* continue;*  *\* }*  *\**  *\* System.out.print(tempNode.value + " <--> ");*  *\* tempNode = tempNode.next;*  *\* }*  *\**  *\* System.out.print("HEAD \n");*  *\*/*      }      public void deleteNodeFromFirst() {          if (head == null) {              System.out.println("Nothing To Delete!");          }          else if (sizeOfLinkedList == 1) {              head = null;              tail = null;              System.out.println("Last Node Deleted. Nothing Else To Delete!");              sizeOfLinkedList--;          } else {              head = head.next;              tail.next = head;              sizeOfLinkedList--;              System.out.println("Node Deleted From First");          }      }      public void deleteNodeFromLast() {          if (head == null) {              System.out.println("Nothing To Delete!");          }          else if (sizeOfLinkedList == 1) {              head = null;              tail = null;              System.out.println("Last Node Deleted. Nothing Else To Delete!");              sizeOfLinkedList--;          } else {              tail = tail.prev;              tail.next = head;              sizeOfLinkedList--;              System.out.println("Node Deleted From Last!");          }      }      public void deleteNode(int *position*) {          if (head == null) {              System.out.println("Nothing To Delete!");          }          else if (sizeOfLinkedList == 1) {              head = null;              tail = null;              System.out.println("Last Node Deleted. Nothing Else To Delete!");              sizeOfLinkedList--;          } else if (*position* == 1) {              deleteNodeFromFirst();          } else if (*position* >= sizeOfLinkedList) {              deleteNodeFromLast();          } else if (*position* <= 0) {              System.out.println("invalid position");          } else {              Node tempNode = head;              for (int i = 1; i < *position* - 1; i++) {                  tempNode = tempNode.next;              }              Node toBeDeleted = tempNode.next;              Node nextNode = tempNode.next.next;              toBeDeleted.prev = null;              toBeDeleted.next = null;              tempNode.next = nextNode;              sizeOfLinkedList--;              System.out.println("Node Deleted From Position " + *position*);          }      }      public int searchNode(String *value*) {          Node tempNode = head;          for (int i = 1; i < sizeOfLinkedList; i++) {              if (tempNode.value == *value*) {                  System.out.println("Node found at position " + i);                  return i;              }          }          System.out.println("Value Not Found in Linked List!");          return -1;      }      public void deleteLinkedList() {          head = null;          tail = null;          sizeOfLinkedList = 0;          System.out.println("Linked List Successfully Deleted!");      }  } |

*main.java*

|  |
| --- |
| public class main {      public static void main(String[] *args*) {          DoublyCircularLinkedList dcll = new DoublyCircularLinkedList();          dcll.insertNodeAtFirst("Cypher");          dcll.insertNodeAtLast("Jett");          dcll.insertNodeAtLast("Sage");          dcll.insertNodeAtLast("Brimstone");          dcll.insertNodeAtLast("Raze");          dcll.insertNodeAtLast("Phoenix");          dcll.insertNodeAtLast("Omen");          dcll.printList();          dcll.deleteNodeFromFirst();          dcll.printList();          System.out.println(dcll.tail.value);          System.out.println(dcll.tail.next.value);          System.out.println(dcll.head.value);          dcll.deleteNodeFromLast();          dcll.printList();          System.out.println(dcll.tail.value);          System.out.println(dcll.tail.next.value);          System.out.println(dcll.head.value);      }  } |

|  |
| --- |
| Implementation Of Doubly Circular Linked Lists |

**Implementation Of Doubly Circular Linked List**

*Node.java*

|  |
| --- |
| public class Node {      String value;      Node next;      Node prev;  } |

*DoublyCircularLinkedList.java*

|  |
| --- |
| public class DoublyCircularLinkedList {      Node head;      Node tail;      int sizeOfLinkedList;  */\* getSizeOfLinkedList() \*/*  */\* createLinkedList() creates a Doubly Linked List \*/*      public void createLinkedList(String *value*) {          Node newNode = new Node();          newNode.value = value;          head = newNode;          tail = newNode;          tail.next = head;          sizeOfLinkedList++;          System.out.println("Linked List is Created!");      }      public int getSizeOfLinkedList() {          return sizeOfLinkedList;      }  */\**  *\* Insert Node*  *\**  *\* 1.Insert Node At First*  *\* 2.Insert Node At Last*  *\* 3.Insert Node At Any Given Position*  *\*/*      public void insertNodeAtFirst(String *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.prev = newNode;              head = newNode;              tail.next = head;              sizeOfLinkedList++;              System.out.println("Node inserted At First!");          }      }  */\* Insert Node At Last! \*/*      public void insertNodeAtLast(String *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.prev = tail;              tail = newNode;              tail.next = head;              sizeOfLinkedList++;              System.out.println("Node inserted At Last!");          }      }  */\* insertNode(String value,int position) Insert Node At Any Given Position \*/*      public void insertNode(String *value*, int *position*) {  */\* Case 1- If linked list is not created before \*/*          if (head == null) {              createLinkedList(value);              return;          }  */\* Case 2- If node is added at the beginning of the linked list \*/*          else if (position == 1) {              insertNodeAtFirst(value);          }  */\* Case 3- If node is added at the end of the linked list \*/*          else if (position > sizeOfLinkedList) {              insertNodeAtLast(value);          }  */\* Case 4- If position is negative or zero. Invalid Case \*/*          else {              insertNodeAtLast(value);          }      }  */\* printList() - Prints the linked list \*/*      public void printList() {          Node tempNode = head;          int count = 1;          while (count != sizeOfLinkedList + 1) {              if (count == sizeOfLinkedList) {                  System.out.print(tempNode.value + " --> ");                  tempNode = tempNode.next;                  count++;                  continue;              }              System.out.print(tempNode.value + " <--> ");              tempNode = tempNode.next;              count++;          }          System.out.print("HEAD \n");  */\**  *\* for (int i = 1; i <= sizeOfLinkedList; i++) {*  *\**  *\* if (i == sizeOfLinkedList) {*  *\* System.out.print(tempNode.value + " --> ");*  *\* tempNode = tempNode.next;*  *\* continue;*  *\* }*  *\**  *\* System.out.print(tempNode.value + " <--> ");*  *\* tempNode = tempNode.next;*  *\* }*  *\**  *\* System.out.print("HEAD \n");*  *\*/*      }      public void deleteNodeFromFirst() {          if (head == null) {              System.out.println("Nothing To Delete!");          }          else if (sizeOfLinkedList == 1) {              head = null;              tail = null;              System.out.println("Last Node Deleted. Nothing Else To Delete!");              sizeOfLinkedList--;          } else {              head = head.next;              tail.next = head;              sizeOfLinkedList--;              System.out.println("Node Deleted From First");          }      }      public void deleteNodeFromLast() {          if (head == null) {              System.out.println("Nothing To Delete!");          }          else if (sizeOfLinkedList == 1) {              head = null;              tail = null;              System.out.println("Last Node Deleted. Nothing Else To Delete!");              sizeOfLinkedList--;          } else {              tail = tail.prev;              tail.next = head;              sizeOfLinkedList--;              System.out.println("Node Deleted From Last!");          }      }      public void deleteNode(int *position*) {          if (head == null) {              System.out.println("Nothing To Delete!");          }          else if (sizeOfLinkedList == 1) {              head = null;              tail = null;              System.out.println("Last Node Deleted. Nothing Else To Delete!");              sizeOfLinkedList--;          } else if (position == 1) {              deleteNodeFromFirst();          } else if (position >= sizeOfLinkedList) {              deleteNodeFromLast();          } else if (position <= 0) {              System.out.println("invalid position");          } else {              Node tempNode = head;              for (int i = 1; i < position - 1; i++) {                  tempNode = tempNode.next;              }              Node toBeDeleted = tempNode.next;              Node nextNode = tempNode.next.next;              toBeDeleted.prev = null;              toBeDeleted.next = null;              tempNode.next = nextNode;              sizeOfLinkedList--;              System.out.println("Node Deleted From Position " + position);          }      }      public int searchNode(String *value*) {          Node tempNode = head;          for (int i = 1; i < sizeOfLinkedList; i++) {              if (tempNode.value == value) {                  System.out.println("Node found at position " + i);                  return i;              }          }          System.out.println("Value Not Found in Linked List!");          return -1;      }      public void deleteLinkedList() {          head = null;          tail = null;          sizeOfLinkedList = 0;          System.out.println("Linked List Successfully Deleted!");      }  } |

*main.java*

|  |
| --- |
| public class main {      public static void main(String[] *args*) {          DoublyCircularLinkedList dcll = new DoublyCircularLinkedList();          dcll.insertNodeAtFirst("Cypher");          dcll.insertNodeAtLast("Jett");          dcll.insertNodeAtLast("Sage");          dcll.insertNodeAtLast("Brimstone");          dcll.insertNodeAtLast("Raze");          dcll.insertNodeAtLast("Phoenix");          dcll.insertNodeAtLast("Omen");          dcll.printList();          dcll.deleteNodeFromFirst();          dcll.printList();          System.out.println(dcll.tail.value);          System.out.println(dcll.tail.next.value);          System.out.println(dcll.head.value);          dcll.deleteNodeFromLast();          dcll.printList();          System.out.println(dcll.tail.value);          System.out.println(dcll.tail.next.value);          System.out.println(dcll.head.value);      }  } |