|  |
| --- |
| Queue |

**Queue**

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

23

25

12

17

18

19

25

27

29

28

Front

Rear

**Implementation Of Queue**

Queue can be implemented in 2 different ways

Implementation Of Queue

Using Arrays

Using Linked List

**Implementation Of Linear Queues Using Arrays**

*Queue.java*

|  |
| --- |
| public class Queue {      int[] arr;      int topOfQueue;      int beginningOfQueue;      public Queue(int *size*) {          arr = new int[*size*];          topOfQueue = -1;          beginningOfQueue = -1;          System.out.println("Queue successfully created with size " + *size*);      }      public boolean isFull() {          if (topOfQueue == arr.length - 1) {              return true;          }          return false;      }      public boolean isEmpty() {          if (topOfQueue == -1 || beginningOfQueue == arr.length) {              return true;          }          return false;      }      public void enqueue(int *value*) {          if (isFull()) {              System.out.println("The Queue Is Full. ");          } else if (isEmpty()) {              beginningOfQueue++;              topOfQueue++;              arr[topOfQueue] = *value*;          } else {              topOfQueue++;              arr[topOfQueue] = *value*;              System.out.println("Element Inserted At Position " + (topOfQueue));          }      }      public int deQueue() {          if (isEmpty()) {              System.out.println("Queue is Empty.Nothing To Return");              return -1;          } else {              int result = arr[beginningOfQueue];              beginningOfQueue++;              if (beginningOfQueue > topOfQueue) {                  topOfQueue = beginningOfQueue = -1;              }              return result;          }      }      public void showQueue() {          for (int i = beginningOfQueue; i <= topOfQueue; i++) {              System.out.print(arr[i] + " ");          }          System.out.println();      }      public int peek() {          if (!isEmpty()) {              return arr[beginningOfQueue + 1];          } else {              System.out.println("Queue is empty!");              return -1;          }      }      public void deleteQueue() {          arr = null;          System.out.println("Queue is Deleted!");      }  } |

*main.java*

|  |
| --- |
| public class main {      public static void main(String[] *args*) {          Queue q1 = new Queue(5);          q1.enqueue(5);          q1.enqueue(6);          q1.enqueue(7);          q1.enqueue(8);          q1.enqueue(9);          q1.showQueue();  *// Overflow*          q1.enqueue(1);          System.out.println(q1.topOfQueue);          q1.deQueue();          q1.showQueue();          System.out.println(q1.beginningOfQueue);          q1.deQueue();          q1.showQueue();          System.out.println(q1.beginningOfQueue);          q1.deQueue();          q1.showQueue();          System.out.println(q1.beginningOfQueue);          q1.deQueue();          q1.showQueue();          System.out.println(q1.beginningOfQueue);          System.out.println(q1.peek());      }  } |

**Implementation Of Queues Using Linked List**

A linear queue when dequeued, leaves a lot of empty cells which cannot be reused. In order to solve this, we need circular queue

*Node.java*

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

*LinkedList.java*

|  |
| --- |
| public class LinkedList {      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 At End \*/*      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");          }      }      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");      }      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!");          }      }      public void deleteLinkedList() {          head = null;          tail = null;          sizeOfLinkedList = 0;          System.out.println("Linked List Deleted!");      }  } |

*Queue.java*

|  |
| --- |
| public class Queue {      int[] arr;      int topOfQueue;      int beginningOfQueue;      public Queue(int *size*) {          arr = new int[*size*];          topOfQueue = -1;          beginningOfQueue = -1;          System.out.println("Queue successfully created with size " + *size*);      }      public boolean isFull() {          if (topOfQueue == arr.length - 1) {              return true;          }          return false;      }      public boolean isEmpty() {          if (topOfQueue == -1 || beginningOfQueue == arr.length) {              return true;          }          return false;      }      public void enqueue(int *value*) {          if (isFull()) {              System.out.println("The Queue Is Full. ");          } else {              arr[topOfQueue + 1] = *value*;              System.out.println("Element Inserted At Position " + (topOfQueue + 1));              topOfQueue++;          }      }      public int deQueue() {          if (isEmpty()) {              System.out.println("Queue is Empty.Nothing To Return");              return 0;          } else {              int result = arr[beginningOfQueue + 1];              beginningOfQueue++;              if (beginningOfQueue == topOfQueue) {                  beginningOfQueue = topOfQueue = -1;              }              return result;          }      }      public void showQueue() {          for (int i = beginningOfQueue + 1; i <= topOfQueue; i++) {              System.out.print(arr[i] + " ");          }          System.out.println();      }      public int peek() {          if (!isEmpty()) {              return arr[beginningOfQueue + 1];          } else {              System.out.println("Queue is empty!");              return -1;          }      }      public void deleteQueue() {          arr = null;          System.out.println("Queue is Deleted!");      }  } |

*main.java*

|  |
| --- |
| public class main {      public static void main(String[] *args*) {          Queue q1 = new Queue();          System.out.println(q1.isEmpty());          q1.enQueue(25);          q1.enQueue(23);          q1.enQueue(27);          q1.enQueue(29);          q1.enQueue(31);          q1.showQueue();          q1.deQueue();          q1.showQueue();          q1.deleteQueue();      }  } |