|  |  |
| --- | --- |
| **Name** | **Hatim Yusuf Sawai** |
| **UID no.** | **2021300108** |
| **Experiment No.** | **2** |

|  |  |
| --- | --- |
| **AIM:** | **Queue Operations Menu-driven program** |
| **Program 1** | |
| **PROBLEM STATEMENT:** | Write a program to make a linear Queue and perform all queue operations on it |
| **THEORY:** | **What is a Queue?**  A queue is an ordered list in which insertions are done at one end (rear) and deletions are done at the other end (front). The first element to be inserted is the first one to be deleted. Hence, it is called First in First out (FIFO). When an element is inserted in a queue, the concept is called EnQueue, and when an element is removed from the queue, the concept is called DeQueue. DeQueueing an empty queue is called underflow and EnQueuing an element in a full queue is called overflow.  **Diagram of Queue:**  **Main Queue Operations:**  **enQueue(int data):** Inserts an element at the end of the queue  **int deQueue():** Removes the element at the front of the queue  **Auxiliary Queue Operations**:  **int Front():** Returns the element at the front without removing it  **int QueueSize():** Returns the no. of elements stored in the queue  **int IsEmptyQueue():** Indicates whether the queue is empty or not |
| **ALGORITHM:** | **Class QueueCheck**  **Main Method:**   1. Input size ‘n’ of queue from user 2. Create an IntQueue object with size ‘n’ 3. While true: 4. Select an Operation from user and store in choice 5. Switch(choice) with cases and default 6. Input whether the user wants to continue 7. If the flag is false then break   **Class IntQueue**  **Members:**   1. Int[] queue 2. Int front,rear,capacity   **Constructor:**   1. Queue = new int[size] 2. Capacity = size 3. front = 0, rear = -1   **Enqueue Method:**   1. if Queue is not full 2. queue[++rear] = e   **Dequeue Method:**   1. if Queue is not empty 2. return queue[front++]   **isEmptyQueue Method:**   1. return front>rear   **isFullQueue Method:**   1. return rear == capacity-1   **front Method:**   1. return queue[front]   **rear Method:**   1. return queue[rear] |
| **PROBLEM SOLVING:** |  |
| **PROGRAM:** | **QueueCheck.java:**  import *java*.*util*.*Scanner*;  import *queueds*.*IntQueue*;  *public* *class* QueueCheck {  *public* *static* void main(String[] args) {          Scanner sc = new Scanner(System.*in*);          System.*out*.print("Enter the size of the queue: ");          int n = sc.nextInt();          IntQueue queue = new IntQueue(n);          int flag,choice;          while(true) {              System.*out*.println("Select an Operation:\n1. Enqueue\t2. Dequeue\n3. Front\t4. Rear\n5. Size");              choice = sc.nextInt();              switch(choice) {                  case 1:                      System.*out*.print("Enter the element to be enqueued: ");                      int e = sc.nextInt();                      try {                          queue.enqueue(e);                          System.*out*.println("Queue: " + queue.printQueue());                      } catch(Exception ex) {                          System.*out*.println(ex.getMessage());                      }                      break;                  case 2:                      try {                          System.*out*.println("Dequeued element: " + queue.dequeue());                          System.*out*.println("Queue: " + queue.printQueue());                      } catch(Exception ex) {                          System.*out*.println(ex.getMessage());                      }                      break;                  case 3:                      try {                          System.*out*.println("Front element: " + queue.front());                      } catch(Exception ex) {                          System.*out*.println(ex.getMessage());                      }                      break;                  case 4:                      try {                          System.*out*.println("Rear element: " + queue.rear());                      } catch(Exception ex) {                          System.*out*.println(ex.getMessage());                      }                      break;                  case 5:                      System.*out*.println("Size: " + queue.size());                      System.*out*.println("Queue: " + queue.printQueue());                      break;                  default:                      System.*out*.println("Invalid choice!");              }              System.*out*.println("Do you want to continue?\n1. Yes\t2. No");              flag = sc.nextInt();              if (flag == 2) {                  break;              }          }          sc.close();      }  }  **IntQueue.java:**  package *queueds*;  *public* *class* IntQueue {      int[] queue;      int front;      int rear;      int capacity;  *public* IntQueue(int size) {          queue = new int[size];          capacity = size;          front = 0;          rear = -1;      }  *public* void enqueue(int e) *throws* Exception {          if(isFullQueue()) {              throw new Exception("Queue is full!");          }          queue[++rear] = e;      }  *public* int dequeue() *throws* Exception {          if(isEmptyQueue()) {              throw new Exception("Queue is empty!");          }          return queue[front++];      }  *public* int front() *throws* Exception {          if(isEmptyQueue()) {              throw new Exception("Queue is empty!");          }          return queue[front];      }  *public* int rear() *throws* Exception {          if(isEmptyQueue()) {              throw new Exception("Queue is empty!");          }          return queue[rear];      }  *public* String printQueue() {          String s = "[";          for(int i=front;i<rear+1;i++) {              s += queue[i]+(i!=rear?",":"");          }          return s += "]";      }  *public* boolean isEmptyQueue() {          return front>rear;      }  *public* boolean isFullQueue() {          return rear == capacity - 1;      }  *public* int size() {          return rear+1-front;      }  } |
| **OUTPUT:**  **Enqueue Operation:**  **QueueFull Check:**  **Dequeue Operation:**  **Queue Empty Check:** | |
| **CONCLUSION:** | In this experiment, we learned how to implement a linear Queue using an array in java and implement the basic operations of a queue data structure. |