**TRIBHUVAN UNIVERSITY**

**JANAPRIYA MULTIPLE CAMPUS**

**Simalchaur,Pokhara-8,Nepal**



**LAB REPORT**

Data Structure and Algorithm with JAVA

Submitted To:

**Mr.Prithvi Raj Paneru**

Department of BIM

Submitted by:

**Krishna Luharuka**

4th Semester

Bachelor of Information Management

Exam Roll No:12435/20

November, 2022

**INDEX:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| SN | Topic | Date of Experiment | Date of submission | Signature | Remark |
| 5. | Operation on Stack.  (linked list and array implementation) |  |  |  |  |
| 6. | Operations on linear queue.(array implementation) |  |  |  |  |
| 7. | Operation on circular queue. (array implementation) |  |  |  |  |
| 8. | Operations on priority queue. (array implementation) |  |  |  |  |

**EXPERIMENT: 5**

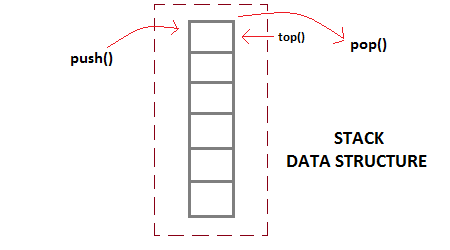
**TITLE:**

A java program to perform operations on stack.

**1.OBJECTIVE:**

* To insert/push a new element into the Stack.
* To remove/pop an element from the top of the Stack.

**2.THEORY:**

Stack is a simple data structure that allows adding and removing elements in a particular order. Every time an element is added, it goes on the top of the stack and the only element that can be removed is the element that is at the top of the stack. It is a LIFO(Last in First out) structure.

The various operations on Stack are:

**2.1. push(x):**

The push(x) operation inserts an item on the top of the stack. If the stack is full then the overflow condition occurs.

**2.2. pop():**

The pop() operation removes an item from the top of the stack. If the stack is empty means that no element exists in the stack, underflow condition occurs.

**2.3. isEmpty():**

It checks whether the stack is empty or not.

.**2.4. isFull():**

The isfull() method checks whether the stack is full or not.

**2.5. display():**

The display() operation traverse to each element and displays the elements of the stack.

**3.IMPLEMENTATION:**

The following programs shows the implementation of stack. There are two ways to implement a stack.

* Using array
* Using linked list

**3.1. Program 1:** Implementing Stack using Arrays:

SOURCE CODE

***class arraystack***

*{*

*int top=-1;*

*int size=4;*

*int a[]=new int[size];*

***public void push(int x)***

*{*

*if(isFull())*

*{*

*System.out.println("Stack Overflow");*

*}*

*else*

*{*

*top++;*

*a[top]=x;*

*}*

*}*

***public int pop()***

*{*

*if(isEmpty())*

*{*

*System.out.println("Stack underflow");*

*return -1;*

*}*

*else*

*{*

*int x=a[top];*

*top--;*

*return x;*

*}*

*}*

***public boolean isEmpty()***

*{*

*if(top==-1)*

*{*

*return true;*

*}*

*else*

*{*

*return false;*

*}*

*}*

***public boolean isFull()***

*{*

*if(top==size-1)*

*{*

*return true;*

*}*

*else*

*{*

*return false;*

*}*

*}*

***public void display()***

*{*

*for(int i=top;i>=0;i--)*

*{*

*System.out.println(a[i]);*

*}*

*System.out.println();*

*}*

***public static void main(String args[])***

*{*

*arraystack as=new arraystack();*

*System.out.println("Stack if it is empty");*

*as.pop();*

*as.display();*

*System.out.println("Stack after push operation");*

*as.push(15);*

*as.push(90);*

*as.push(5);*

*as.push(9);*

*as.display();*

*System.out.println("Stack if it is full");*

*as.push(6);*

*System.out.println();*

*System.out.println("Stack after pop operation of two items");*

*as.pop();*

*as.pop();*

*as.display();*

*}*

*}*

OUTPUT:

Stack if it is empty

Stack underflow

Stack after push operation

9

5

90

15

Stack if it is full

Stack Overflow

Stack after pop operation of two items

90

15

**3.2. Program 2:** Implementing Stack using Linked List:

SOURCE CODE:

***class liststack***

*{*

***class node***

*{*

*int data;*

*node next;*

*node(int data)*

*{*

*this.data=data;*

*this.next=null;*

*}*

*}*

*public node top=null;*

***public void push(int x)***

*{*

*node newnode=new node(x);*

*if(top==null)*

*{*

*top=newnode;*

*}*

*else*

*{*

*newnode.next=top;*

*top=newnode;*

*}*

*}*

***public int pop()***

*{*

*if(isEmpty())*

*{*

*System.out.println("Stack underflow");*

*return -1;*

*}*

*else*

*{*

*int x=top.data;*

*top=top.next;*

*return x;}*

*}*

***public boolean isEmpty()***

*{*

*if(top==null)*

*{*

*return true;*

*}*

*else*

*return false;*

*}*

***public void display()***

*{*

*node temp=top;*

*while(temp!=null)*

*{*

*System.out.println(temp.data);*

*temp=temp.next;*

*}*

*}*

***public static void main(String args[])***

*{*

*liststack as=new liststack();*

*System.out.println("Stack if it is empty");*

*as.pop();*

*as.display();*

*System.out.println();*

*System.out.println("Stack after push operation");*

*as.push(15);*

*as.push(90);*

*as.push(5);*

*as.push(9);*

*as.display();*

*System.out.println();*

*System.out.println("Stack after pop operation of two items");*

*as.pop();*

*as.pop();*

*as.display();*

*}*

*}*

OUTPUT:

Stack if it is empty

Stack underflow

Stack after push operation

9

5

90

15

Stack after pop operation of two items

90

15

**4. OUTPUT AND DISCUSSION:**

* In this experiment, the output of program 1 shows stack underflow state when there is no element in the stack. The push() method pushes/adds the element to the stack. Similarly, the pop() operation removes the elements and the remaining element of the stack are displayed. Moreover, when the stack was full it displayed stack overflow state.
* The output of program 2 shows the similar operation as above using linked list. Besides, as the linked list has no size limitation isfull() method is not used and it doesn’t have stack overflow state.

**5. CONCLUSION:**

A Java program to insert and remove element in and from the stack is successfully run.

**EXPERIMENT: 6**

**TITLE:**

A java program to perform operations on linear queue.

**1.OBJECTIVE:**

* To insert/enqueue a new element into the Queue.
* To remove/dequeue an element from the front of the queue.

**2.THEORY:**

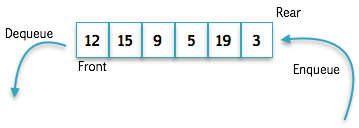
A linear queue is a linear data structure that serves the request first, which has been arrived first. It consists of data elements which are connected in a linear fashion. It has two pointers i.e, front and rear where the insertion takes place from the front end and deletion occurs from the front end.

Fig: Linear queue

The various operations on linear queue are:

**2.1. Enqueue(x):**

The enqueue operation inserts the new element from the rear end.

**2.2. Dequeue():**

The dequeue operation is used to delete the existing element from the front end of the queue.

**2.3. isEmpty():**

It checks whether the queue is empty or not. It returns true if the queue is empty otherwise false.

.**2.4. isFull():**

The isfull() method checks whether the queue is full or not. It returns true if the queue is full otherwise return false.

**2.5. display():**

The display() operation traverse to each element and displays the elements of the queue.

**3.IMPLEMENTATION:**

***class linearqueue***

*{*

*int size=4;*

*int currsize=0;*

*int array[]=new int[size];*

*int front=0;*

*int rear=-1;*

***public void enqueue(int x)***

*{*

*if(isFull()){*

*System.out.println("Queue Overflow");*

*}*

*if(rear==size-1)*

*{*

*rear=-1;*

*}*

*rear++;*

*array[rear]=x;*

*currsize++;*

*}*

***public int dequeue()***

*{*

*int x=-1;*

*if(isEmpty()){*

*System.out.println("Queue underflow");*

*return x;*

*}*

*else {*

*x=array[front];*

*front++;*

*if(front==size){*

*front=0;*

*}*

*currsize--;*

*return x;*

*}*

*}*

***public boolean isFull()***

*{*

*if(currsize==size)*

*return true;*

*else*

*return false;*

*}*

***public boolean isEmpty()***

*{*

*if(currsize==0)*

*return true;*

*else*

*return false;*

*}*

***public void display()***

*{*

*for(int i=front;i<=rear;i++)*

*{*

*System.out.print(array[i]+" ");*

*}*

*System.out.println();*

*}*

***public static void main(String args[])***

*{*

*linearqueue q=new linearqueue();*

*System.out.println("if the linear queue is empty");*

*q.dequeue();*

*q.display();*

*System.out.println("if 4 elements are enqueued in the queue.");*

*q.enqueue(5);*

*q.display();*

*q.enqueue(10);*

*q.display();*

*q.enqueue(15);*

*q.display();*

*q.enqueue(20);*

*q.display();*

*System.out.println("if the queue is full and 2 elements are enqueued");*

*q.enqueue(5);*

*q.display();*

*q.enqueue(15);*

*q.display();*

*System.out.println("If an element is dequeued");*

*q.dequeue();*

*q.display();*

*}*

*}*

OUTPUT:

if the linear queue is empty

Queue underflow

if 4 elements are enqueued in the queue.

5

5 10

5 10 15

5 10 15 20

if the queue is full and 2 elements are enqueued

Queue Overflow

5

5 15

If an element is dequeued

15

**OUTPUT AND DISCUSSION:**

In the above implementation, it is seen that when the dequeue operation is run when there no element enqueued, it displays queue underflow. Then, the elements are enqueued using enqueue() operation. When the size of the queue is full, the output displays queue overflow. Finally, the element is removed from the queue using dequeue operation.

**CONCLUSION:**

A Java program to enqueue and dequeue element in and from the linear queue is successfully run.

**EXPERIMENT: 7**

**TITLE:**

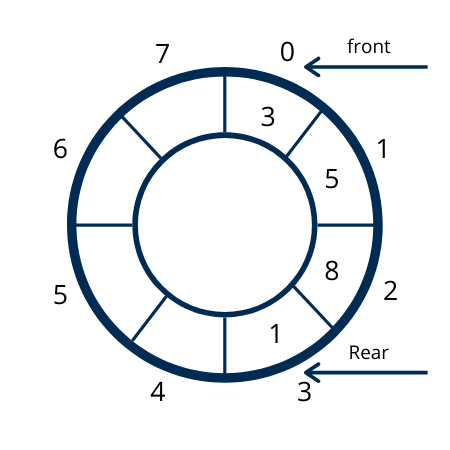
A java program to perform operations on circular queue.

**1.OBJECTIVE:**

* To insert/enqueue a new element into the circular queue.
* To remove/dequeue an element from the front of the queue.

**2.THEORY:**

A circular queue is also a linear data structure like a normal queue that follows the FIFO principle but it does not end the queue; it connects the last position of the queue to the first position of the queue. If a new element is to be inserted at the beginning, it is done using circular queue data structure. The operations are performed based on FIFO (First In First Out) principle. It is also called **‘Ring Buffer’**.



**Fig: Circular Queue**

The various operations on circular queue are:

**2.1. Enqueue(x):**

The enqueue operation inserts the new element in the queue. If the queue is empty, then the front and rear are set to 0 to insert a new element. If queue is not empty, then value of the rear gets incremented. If queue is not empty and rear is equal to n-1, then rear is set to 0.

**2.2. Dequeue():**

This function is used to delete an element from the circular queue. In a circular queue, the element is always deleted from front position. It first checks whether queue is Empty means check (front==-1) and if it is empty then display Queue is empty. If queue is not empty then checks if (front==rear) if it is true then set front=rear= -1 else check if (front==size-1), if it is true then set front=0 and return the element.

**2.3. isEmpty():**

It checks whether the queue is empty or not. It returns true if the queue is empty otherwise false.

.**2.4. isFull():**

The isfull() method checks whether the queue is full or not. It returns true if the queue is full otherwise return false.

**2.5. display():**

The display() operation traverse to each element and displays the elements of the queue.

**3.IMPLEMENTATION:**

For the implementation of a circular queue using array, initialize an array queue of size n, where n is the maximum number of elements that the queue can hold also initialize two variables front and rear to -1. Then to enqueue an element x onto the queue, increase rear by 1 If rear is equal to n, set rear to 0. If front is -1, set front to 0. Then, Set queue[rear] to x.

Similarly, To dequeue an element from the queue, Check if the queue is empty by checking if front is -1. If it is, return an error message indicating that the queue is empty. Set x to queue[front]. If front is equal to rear, set front and rear to -1. Otherwise, increment front by 1 and if front is equal to n, set front to 0. Return x.

SOURCE CODE:

***class circularqueue***

*{*

*int size=4;*

*int front=-1;*

*int rear=-1;*

*int array[]=new int[size];*

*public void enqueue(int x)*

*{*

*if(isFull())*

*{*

*System.out.println("QUEUE OVERFLOW");*

*}*

*if(front==-1)*

*front=0;*

*rear=(rear+1)%size;*

*array[rear]=x;*

*}*

***public int dequeue()***

*{*

*if(isEmpty())*

*{*

*System.out.println("QUEUE UNDERFLOW");*

*return -1;*

*}*

*else*

*{*

*int x=array[front];*

*if(front==rear)*

*{*

*front=-1;*

*rear=-1;*

*}*

*else*

*{*

*front=(front+1)%size;*

*}*

*return x;*

*}*

*}*

***public boolean isFull()***

*{*

*if(front==0 && rear==size-1)*

*return true;*

*if(front==rear+1)*

*return true;*

*else*

*return false;*

*}*

***public boolean isEmpty()***

*{*

*if(front==-1)*

*return true;*

*else*

*return false;*

*}*

***public void display()***

*{*

*if(isEmpty())*

*{*

*System.out.println("queue underflow");*

*}*

*else*

*{*

*if(front==rear)*

*{*

*System.out.println(array[front]);*

*}*

*else*

*{*

*for(int i=front;i!=rear;i=(i+1)%size)*

*{*

*System.out.print(array[i]+" ");*

*}*

*System.out.println(array[rear]);*

*}*

*}*

*}*

***public static void main(String args[])***

*{*

*circularqueue q=new circularqueue();*

*System.out.println("If the queue is empty");*

*q.dequeue();*

*System.out.println("If 2 elements are enqueud");*

*q.enqueue(5);*

*q.display();*

*q.enqueue(10);*

*q.display();*

*System.out.println("if a element is dequeue");*

*q.dequeue();*

*q.display();*

*System.out.println("if three element is enqueue");*

*q.enqueue(35);*

*q.display();*

*q.enqueue(45);*

*q.display();*

*q.enqueue(55);*

*q.display();*

*System.out.println("If 1 element is enqueued to the queue");*

*q.enqueue(56);*

*q.display();*

*}*

*}*

OUTPUT:

If the queue is empty

QUEUE UNDERFLOW

If 2 elements are enqueued

5

5 10

if an element is dequeue

10

if three element is enqueue

10 35

10 35 45

10 35 45 55

If 1 element is enqueued to the queue

QUEUE OVERFLOW

56

**4.OUTPUT AND DISCUSSION:**

In the above implementation, it is seen that when the dequeue operation is run when there is no element enqueued, it displays queue underflow. Then, the elements are enqueued using enqueue() operation and after a element is dequeued using dequeue() method. Again the elements are enqueued until the queue is full. When the size of the queue is full, the output displays queue overflow.

**5.CONCLUSION:**

A Java program to enqueue and dequeue element in and from the circular queue is successfully run.

**EXPERIMENT: 8**

**TITLE:**

A java program to perform operations on priority queue.

**1.OBJECTIVE:**

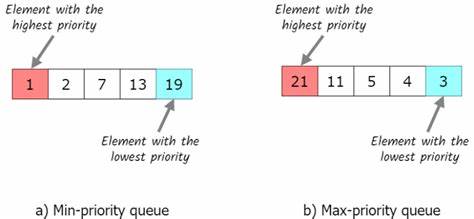
* To find the index of maximum or minimum element in the queue.
* To insert/enqueue a new element into the queue.
* To remove/dequeue the maximum or minimum element from the queue.

**2.THEORY:**

A priority queue is a collection of elements such that each element has been assigned a priority value such that the order in which the elements are processed comes from the following rules:

* An element of higher priority is processed before any element of lower priority.
* Element with same order follow FIFO principle.

Two types of priority queue are:

* **Max-Priority Queue**: In this, maximum value element is removed first from the queue.
* **Min-Priority Queue**: In this, minimum value element is removed first from the queue.

The various operations on priority queue are:

**2.1. insert():**

This operation inserts a value at queue.

**2.2. remove():**

It removes the maximum/minimum value from the queue.

**2.3. findmax():**

It finds the index of maximum value in the queue.

**2.4. findmin():**

It finds the index of minimum value in the queue and return it.

**2.5. isEmpty():**

It checks whether the queue is empty or not. It returns true if the queue is empty otherwise false.

.**2.6. isFull():**

The isfull() method checks whether the queue is full or not. It returns true if the queue is full otherwise return false.

**2.5. display():**

The display() operation traverse to each element and displays the elements of the queue.

**3. IMPLEMENTATION:**

The following source code is for the implementation of the min-priority queue. In the min-priority queue, elements are inserted according to their arrival order and the smallest element is deleted first from max priority queue.  
 SOURCE CODE*:*

***class priorityqueue***

*{*

*int rear=-1;*

*int size=4;*

*int array[]=new int[size];*

*int currsize=0;*

*int maxsize=size;*

***public int findMin()***

*{*

*if(isEmpty())*

*{*

*return -1;*

*}*

*else*

*{*

*int min=0;*

*for(int i=1;i<=currsize;i++)*

*{*

*if(array[i]<array[min])*

*min=i;*

*}*

*return min;*

*}*

*}*

***public void enqueue(int x)***

*{*

*if(isFull())*

*{*

*System.out.println("Queue Overflow");*

*}*

*if(rear==size-1)*

*{*

*rear=-1;*

*}*

*rear++;*

*array[rear]=x;*

*currsize++;*

*}*

***public int dequeue()***

*{*

*if(isEmpty())*

*{*

*System.out.println("Queue underflow");*

*return -1;*

*}*

*else*

*{*

*int min=findMin();*

*int x=array[min];*

*for(int i=min;i<=rear;i++)*

*{*

*array[i]=array[i+1];*

*}*

*rear--;*

*return x;*

*}*

*}*

***public boolean isFull()***

*{*

*if(rear==maxsize-1)*

*return true;*

*else*

*return false;*

*}*

***public boolean isEmpty()***

*{*

*if(rear==-1)*

*return true;*

*else*

*return false;*

*}*

*public void display()*

*{*

*for(int i=0;i<=rear;i++)*

*{*

*System.out.print(array[i]+" ");*

*}*

*System.out.println();*

*}*

***public static void main(String args[])***

*{*

*priorityqueue q=new priorityqueue();*

*System.out.println("If the queue is empty");*

*q.dequeue();*

*q.display();*

*System.out.println("If 3 elements are inserted in the queue");*

*q.enqueue(15);*

*q.display();*

*q.enqueue(10);*

*q.display();*

*q.enqueue(5);*

*q.display();*

*System.out.println();*

*System.out.println("If a element is removed from the queue");*

*q.dequeue();*

*q.display();*

*System.out.println();*

*System.out.println("If 2 more elements are enqueued");*

*q.enqueue(95);*

*q.display();*

*q.enqueue(85);*

*q.display();*

*System.out.println();*

*System.out.println("If the queue is full and a element is inserted");*

*q.enqueue(75);*

*q.display();*

OUTPUT:

If the queue is empty

Queue underflow

If 3 elements are inserted in the queue

15

15 10

15 10 5

If a element is removed from the queue

15 10

If 2 more elements are enqueued

15 10 95

15 10 95 85

If the queue is full and a element is inserted

Queue Overflow

75

**4.OUTPUT AND DISCUSSION:**

In the above implementation, it is seen that when the dequeue operation is run when there no element enqueued, it displays queue underflow. Then, the elements are enqueued using enqueue() operation. When the size of the queue is full, the output displays queue overflow. Finally, the element is removed from the queue using findmin() and dequeue() operation.

**5.CONCLUSION:**

A Java program to enqueue and dequeue element in and from the priority queue is successfully run.