

IT623 - Lab Assignment 5

1. Create a class called **IntQueue**, implement an integer queue using an array.

Code:

```
class Program1 {
    int size = 5;
    int queue[] = new int[size];
    int front, rear;

    public Program1() {
        front = -1;
        rear = -1;
    }

    boolean empty() {
        if (front == -1)
            return true;
        else
            return false;
    }

    void enqueue(int x) {
        if (front == 0 && rear == size - 1)
            System.out.println("Queue is Full!!");
        else {
            if (front == -1)
                front = 0;
            rear++;
        }
    }
}
```

```
        queue[rear] = x;
        System.out.println("Inserted element is " + x);
    }
}

void dequeue() {
    int x;
    if (empty())
        System.out.println("Queue is Empty!!");
    else {
        x = queue[front];
        if (front >= rear) {
            front = -1;
            rear = -1;
        } else {
            front++;
        }
        System.out.println("Deleted -> "+x);
    }
}

void front() {
    if (front == rear) {
        System.out.println("Queue is Empty");
        return;
    }
    System.out.println("\nFront element : " + queue[front]);
    return;
}

void display() {
    if (empty()) {
        System.out.println("Queue is Empty!!");
    }
}
```

```
        } else {  
  
            System.out.print("Elements -> ");  
            for (int i = front; i <= rear; i++)  
                System.out.print(queue[i] + " ");  
  
        }  
    }  
  
    public static void main(String[] args) {  
        Program1 p = new Program1();  
  
        System.out.println("Is empty : "+p.empty());  
        p.enqueue(1);  
        p.enqueue(2);  
        p.enqueue(3);  
        p.enqueue(4);  
        p.enqueue(5);  
        p.enqueue(6);  
  
        p.display();  
  
        p.front();  
  
        p.dequeue();  
        p.display();  
    }  
}
```

Output Snapshot:

```

<terminated> Program1 (3) [Java Application] C:\Program Files\Java\jre1.8.0_241\bin\javaw.exe (18-Oct-2021, 9:29:29 am)
Is empty : true
Inserted element is 1
Inserted element is 2
Inserted element is 3
Inserted element is 4
Inserted element is 5
Queue is Full!!
Elements -> 1 2 3 4 5
Front element : 1
Deleted -> 1
Elements -> 2 3 4 5

```

2. Implement Circular Queue.

Code:

```

class CircularQueue {
    int size = 10;
    int front, rear;
    int items[] = new int[size];

    CircularQueue() {
        front = -1;
        rear = -1;
    }

    boolean empty() {
        if (front == -1) {
            return true;
        } else {
            return false;
        }
    }

    void enqueue(int data) {
        if (front == 0 && rear == size - 1 || front == rear + 1) {
            System.out.print("Queue is full!!");

```

```
    } else {  
        if (front == -1)  
            front = 0;  
    }  
    rear = (rear + 1) % size;  
    items[rear] = data;  
}
```

```
int dequeue() {  
    int data;  
    if (front == -1) {  
        System.out.println("Queue is empty!!");  
        return (-1);  
    } else {  
        data = items[front];  
        if (front == rear) {  
            front = -1;  
            rear = -1;  
        } else {  
            front = (front + 1) % size;  
        }  
        return data;  
    }  
}
```

```
void front() {  
    if (front == -1) {  
        System.out.println("Queue is Empty!!");  
        return;  
    }  
    System.out.println("\nFront element : " + items[front]);  
    return;  
}
```

```
void display() {
    if (front == -1) {
        System.out.println("Queue is empty");
    } else {
        for (int i = front; i <= rear; i++) {
            System.out.print(items[i] + " ");
        }
    }
}

}

public class Program2 {
    public static void main(String args[]) {
        CircularQueue cq = new CircularQueue();

        System.out.println("Queue is empty? : " + cq.empty());

        System.out.print("Circular queue after insertion : ");
        cq.enqueue(1);
        cq.enqueue(2);
        cq.enqueue(3);
        cq.enqueue(4);
        cq.enqueue(5);

        cq.display();

        int data = cq.dequeue();

        if (data != -1) {
            System.out.println("\nDeleted element : " + data);
        }

        System.out.print("Circular queue after deletion : ");
```

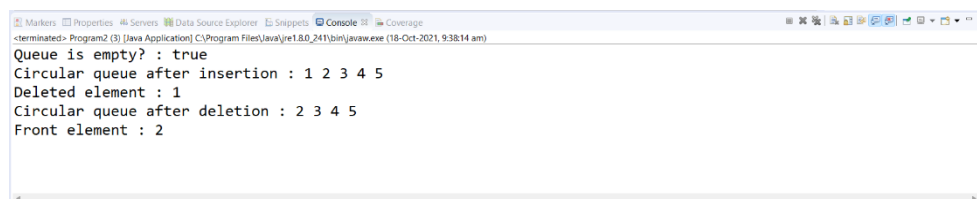
```

        cq.display();

        cq.front();
    }
}

```

Output Snapshot:



```

<terminated> Program2 (3) [Java Application] C:\Program Files\Java\jre1.8.0_241\bin\javaw.exe (18-Oct-2021, 9:38:14 am)
Queue is empty? : true
Circular queue after insertion : 1 2 3 4 5
Deleted element : 1
Circular queue after deletion : 2 3 4 5
Front element : 2

```

3. Implement Cache using Queues of size 3 and return the number of page faults.

Code:

```

class PageFaultQueue {
    int front, rear, size, flag = 0;
    int capacity;
    int queue[];
    int pagefault = 0;

    public PageFaultQueue(int size) {
        capacity = size;
        front = -1;
        rear = -1;
        queue = new int[capacity + 1];
    }
}

```

```
boolean empty() {
    if (front == -1 || rear == -1) {
        return true;
    }
    return false;
}

public void enqueue(int item) {
    if (front == -1 && rear == -1) {
        front = 0;
        rear = 0;
        queue[rear] = item;
        pagefault++;
    } else if ((rear + 1) % capacity == front) {
        rear = -1;
        for (int i = 0; i < 3; i++) {
            if (item == queue[i]) {
                flag = 1;
                break;
            }
        }
        if (flag == 1) {
            flag = 0;
        } else {
            dequeue();
            rear++;
            queue[rear] = item;
            pagefault++;
        }
    } else {
        rear = (rear + 1) % capacity;
        for (int i = 0; i < 3; i++) {
            if (item == queue[i]) {
                flag = 1;
```



```
                break;
            }
        }
        if (flag == 1) {
            flag = 0;
        } else {
            queue[rear] = item;
            pagefault++;
        }
    }
}

public int dequeue() {
    int item = 0;
    if (front == -1 && rear == -1) {
        System.out.println("Underflow");
    } else if (front == rear) {
        item = queue[front];
        front = -1;
        rear = -1;
    } else {
        item = queue[front];
        front = (front + 1) % capacity;
    }
    return item;
}

void display() {
    System.out.println("Total Page fault -> " + pagefault);
}

}
```

```
public class Program3 {  
    public static void main(String[] args) {  
        PageFaultQueue p = new PageFaultQueue(3);  
  
        p.enqueue(1);  
  
        p.enqueue(2);  
        p.enqueue(3);  
        p.enqueue(4);  
        p.enqueue(2);  
        p.enqueue(5);  
        p.enqueue(1);  
  
        p.display();  
    }  
}
```

Output Snapshot:



- 4. Implement stack using queues(do not change queue functionalities i.e.- remove the item from the front only not from the rear).**

Code:

```
class Queue {  
    int front, rear;  
    int queue[];
```

```
int size;
```

```
Queue(int s) {  
    size = s;  
    queue = new int[size];  
    front = rear = -1;  
}
```

```
void enqueue(int data) {  
    if (front == -1 && rear == -1) {  
        front = rear = 0;  
        queue[rear] = data;  
    } else if ((rear + 1) == size) {  
        System.out.println("Overflow");  
    } else {  
        queue[++rear] = data;  
    }  
}
```

```
int dequeue() {  
    if (front + 1 > rear) {  
        int x = queue[front];  
        front = rear = -1;  
        return x;  
    }  
    return queue[front++];  
}
```

```
int peek() {  
    return queue[front];  
}
```

```
    }

    boolean empty() {
        return (front == -1 && rear == -1);
    }
}

class Stack {
    Queue q1;
    int size;

    Stack(int s) {
        this.size = s;
        q1 = new Queue(s);
    }

    void push(int data) {
        Queue q2 = new Queue(this.size);
        while (!q1.empty()) {
            q2.enqueue(q1.dequeue());
        }
        q1.enqueue(data);
        while (!q2.empty()) {
            q1.enqueue(q2.dequeue());
        }
    }

    int peek() {
        return q1.peek();
    }

    int pop() {
        return q1.dequeue();
    }
}
```

```
        boolean empty() {
            return q1.empty();
        }
    }

    public class Program4 {
        public static void main(String[] args) {
            Stack s1 = new Stack(7);

            System.out.println("Is stack empty ? : " + s1.empty());
            s1.push(1);
            s1.push(2);
            s1.push(3);

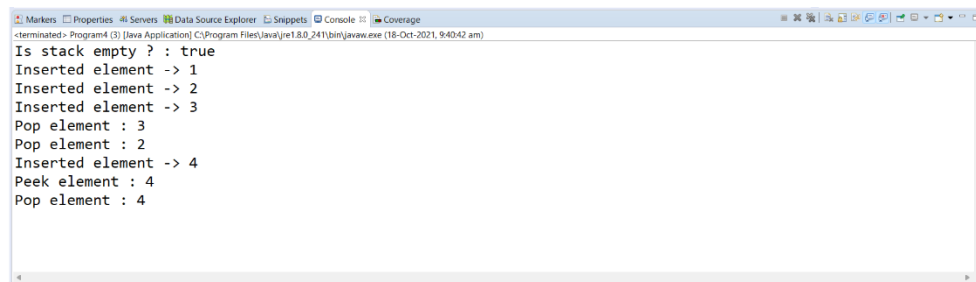
            if (!s1.empty()) {
                System.out.println("Pop element : " + s1.pop());
            }

            if (!s1.empty()) {
                System.out.println("Pop element : " + s1.pop());
            }

            s1.push(4);
            System.out.println("Peek element : " + s1.peek());

            if (!s1.empty()) {
                System.out.println("Pop element : " + s1.pop());
            }
        }
    }
```

Output Snapshot:

A screenshot of a Java IDE's console window. The window title is "Console" and it shows the output of a Java program. The output text is as follows:

```
<terminated> Program4 (3) [Java Application] C:\Program Files\Java\jre1.8.0_241\bin\javaw.exe (18-Oct-2021, 9:40:42 am)
Is stack empty ? : true
Inserted element -> 1
Inserted element -> 2
Inserted element -> 3
Pop element : 3
Pop element : 2
Inserted element -> 4
Peek element : 4
Pop element : 4
```

5. In the last lab, we implemented the stack. Now we have stack and queue. Our task is to compute an alternating series using the numbers that you entered. An alternating series switches the sign of the number being added for every other number. For example, if we enter the numbers: 1, 3, 15, 9 then we would compute the alternating series as $1 - 3 + 15 - 9 = 4$. Write the Queue code so that it computes the sum of an alternating series of the numbers as they are removed from the queue (so the first number is positive, the second is negative, etc.) in the deque function. Display the alternating series to the screen on a single line. So if the numbers 1, 3, 15, and 9 were entered by the user, the code should display: $1 - 3 + 15 - 9 = 4$.

Code:

```
class Program5 {
    int size;
    int queue[];
    int front, rear, sum = 0;
```

```
public Program5(int s) {
    queue = new int[s];
    size = s;
    front = -1;
    rear = -1;
}

boolean empty() {
    if (front == -1)
        return true;
    else
        return false;
}

void enqueue(int x) {
    if (front == 0 && rear == size - 1)
        System.out.println("Queue is Full!!");
    else {
        if (front == -1)
            front = 0;
        rear++;
        queue[rear] = x;
    }
}

void dequeue() {
    int x = 0;

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

        if (empty())
```

```
        System.out.println("Queue is Empty!!");
    else {
        x = queue[front];
        if (front >= rear) {
            front = -1;
            rear = -1;
        } else {
            front++;
        }
    }

    if (i % 2 == 0) {
        sum += x;
        if (rear == -1) {
            System.out.print(x);
            break;
        }
        System.out.print(x + " - ");
    } else {
        sum -= x;
        if (rear == -1) {
            System.out.print(x);
            break;
        }
        System.out.print(x + " + ");
    }
}

System.out.print(" = " + sum);
}

public static void main(String[] args) {
```



```
Program5 p = new Program5(5);

p.enqueue(1);
p.enqueue(3);
p.enqueue(15);
p.enqueue(9);

p.dequeue();
}
```

Output Snapshot:

