**Queue**

***Queue Using Array***

public class QueueUsingArray {

private int[] data;

private int front;

private int rear;

private int size;

public QueueUsingArray(int capacity) {

data = new int[capacity];

front = -1;

rear = -1;

}

public QueueUsingArray() {

data = new int[5];

front = -1;

rear = -1;

}

// 1st method

public void enqueue(int element) {

if (size == data.length) {

doubleCapacity();

}

if (size == 0) {

front++;

}

data[rear] = element;

size++;

}

// Helper method

private void doubleCapacity() {

int[] temp = data;

data = new int[2 \* temp.length];

int index = 0;

for (int i = front; i < temp.length; i++) {

data[index++] = temp[i];

}

for (int i = 0; i < front - 1; i++) {

data[index++] = temp[i];

}

front = 0;

rear = temp.length - 1;

}

// 2nd method

public int size() {

return size;

}

// 3rd method

public boolean isEmpty() {

return size == 0;

}

// 4th method

public int front() {

if (size == 0) {

return -1;

}

return data[front];

}

// 5th method

public int dequeue() {

if (size == 0) {

return -1;

}

int temp = data[front];

front = (front + 1) % data.length;

size--;

if (size == 0) {

front = -1;

rear = -1;

}

return temp;

}

}

***Queue Using Linked List***

public class Node<T> {

public T data;

public Node<T> next;

public Node(T data) {

this.data = data;

}

}

public class QueueUsingLL<T> {

private int size;

private Node<T> front;

private Node<T> rear;

public QueueUsingLL() {

front = null;

rear = null;

size = 0;

}

// 1st method

public int size() {

return size;

}

// 2nd method

public boolean isEmpty() {

return size == 0;

}

// 3rd method

public void enqueue(T ele) {

Node<T> newNode = new Node<>(ele);

size++;

if (front == null) {

front = newNode;

rear = newNode;

} else {

rear.next = newNode;

rear = newNode; //or rear=rear.next;

}

}

// 4th method

public T front(){

if (front == null){

return;

}

return front.data;

}

// 5th method

public T dequeue(){

if (front == null) {

return; }

T temp = front.data;

front = front.next;

if (front == null) {

rear = null;

}

size--;

return temp;

}

}

***Queue Using Stack***

public class QueueUsingStack<T> {

Stack<T> stack1;

Stack<T> stack2;

int size = 0;

public QueueUsingStack() {

stack1 = new Stack<>();

stack2 = new Stack<>();

}

// 1st method

public void enqueue(T ele) {

stack1.push(ele);

size++;

}

// 2nd method

public T front() {

while (!stack1.isEmpty())

stack2.push(stack1.pop());

T temp = stack2.peek();

while (!stack2.isEmpty())

stack1.push(stack2.pop());

return temp;

}

// 3rd method

public T rear() {

return stack1.peek();

}

// 4th method

public int size() {

return size;

}

// 5th method

public boolean isEmpty() {

return size == 0;

}

// 6th method

public T dequeue() {

if (stack1.isEmpty())

return null;

while (!stack1.isEmpty())

stack2.push(stack1.pop());

T temp = stack2.pop();

while (!stack2.isEmpty())

stack1.push(stack2.pop());

size--;

return temp;

}

}