**BuildQueuingclasses**

1. Write 2 classes to implement the following IQueuable interface Interface IQueuable {  
   //adds value to queue and returns new queue enqueue(value: string): string[];

//removes item from queue, and returns the item removed dequeue(): string;

//returns a list of all the items in the queue getQueue(): string[];

//returns the number of items in the queue size():number;  
}

Build your queues ontop of arrays; call your classes FIFOQUEUE and LIFOQUEUE (first-in first-out & last-in first-out) queues, or QUEUE and STACK, whichever names you prefer.

1. If you’ve written your enqueue and dequeue using array methods, how would you rewrite the same functions without using any array methods?
2. Show how you would improve the design of your classes, apply various design patterns and techniques as you see fit

**Question 1.**

class Queue implements IQueuable{

private int front, rear, size;

private String queue[];

Queue(int capacity) {

this.front = 0;

this.rear = 0;

this.size = capacity;

this.queue = new String[capacity];

}

String[] enqueue(String item) {

// if end of queue is equal to size of array queue, the queue is full

if (rear==size) {

System.out.println("Queue is full");

}

// insert element at the rear of the queue

else {

queue[rear] = item;

// move rear by one index

rear++;

}

return queue;

}

String dequeue() {

//removedItem set to first item in queue

String removedItem=queue[0];

// if front of queue is equal to the end of queue, the queue is empty

if (front == rear) {

System.out.println("Queue is empty");

}

// remove item from queue by shifting the array by one index

else {

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

queue[i] = queue[i + 1];

}

// decrement rear

rear--;

}

return removedItem;

}

// queue elements

String[] getQueue()

{

return queue;

}

int size(){

int count=0;

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

count++;

}

return count;

}

}

class Stack implements IQueuable{

private int top, size;

private String queue[];

Stack(int capacity) {

this.top = 0;

this.size = capacity;

this.queue = new String[capacity];

}

String[] enqueue(String item) {

// if top of stack is equal to size of array queue, the stack is full

if (top==size) {

System.out.println("Stack is full");

}

// insert element at the top

else {

queue[top] = item;

// move top by one index

top++;

}

return queue;

}

String dequeue() {

//decrement top

top--;

//set String to removedItem

String removedItem=queue[top];

// if top of queue is equal -1, or one less than zero, the queue is empty

if (top == -1) {

System.out.println("Stack is empty");

}

return removedItem;

}

// queue elements

String[] getQueue()

{

return queue;

}

int size(){

int count=0;

for (int i = 0; i < top; i++) {

count++;

}

return count;

}

}

**Question 2.**

I have not utilised any array methods in creating the enqueue and dequeue methods.

**Question 3.**

There are several ways that we can improve the design of our classes;

Encapsulation: We can improve encapsulation by making the data structures private and providing public methods to access them. This will make the code more modular and maintainable.

Abstraction: We can abstract away the implementation details of the data structures by creating an abstract class or interface for the queue. This will allow us to easily switch between different types of queues without changing the rest of the code.

Observer Pattern: We can use the observer pattern to notify the client code when an item is added or removed from the queue. This will allow the client code to react to changes in the queue.