TypeScript Implementation

FIFO Queue

interface IQueueable {

enqueue(value: string): string[];

dequeue(): string;

getQueue(): string[];

size(): number;

}

class FIFOQueue implements IQueueable {

private queue: string[] = [];

enqueue(value: string): string[] {

this.queue.push(value);

return this.queue;

}

dequeue(): string {

return this.queue.shift();

}

getQueue(): string[] {

return this.queue;

}

size(): number {

return this.queue.length;

}

}

LIFO Queue

class LIFOQueue implements IQueueable {

private queue: string[] = [];

enqueue(value: string): string[] {

this.queue.push(value);

return this.queue;

}

dequeue(): string {

return this.queue.pop();

}

getQueue(): string[] {

return this.queue;

}

size(): number {

return this.queue.length;

}

}

A. b. Rewrite Functions without Array Methods

interface IQueueable {

enqueue(value: string): string[];

dequeue(): string;

getQueue(): string[];

size(): number;

}

class FIFOQueue implements IQueueable {

private queue: string[] = [];

enqueue(value: string): string[] {

this.queue[this.queue.length] = value;

return this.queue;

}

dequeue(): string {

let item = this.queue[0];

for (let i = 0; i < this.size(); i++) {

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

}

this.queue.length--;

return item;

}

getQueue(): string[] {

let queue: string[] = [];

for (let i = 0; i < this.size(); i++) {

queue[i] = this.queue[i];

}

return queue;

}

size(): number {

let counter = 0;

for (let i in this.queue) {

counter++;

}

return counter;

}

}

class LIFOQueue implements IQueueable {

private queue: string[] = [];

enqueue(value: string): string[] {

this.queue[this.queue.length] = value;

return this.queue;

}

dequeue(): string {

let item = this.queue[this.size() - 1];

this.queue.length--;

return item;

}

getQueue(): string[] {

let queue: string[] = [];

for (let i = 0; i < this.size(); i++) {

queue[i] = this.queue[i];

}

return queue;

}

size(): number {

let counter = 0;

for (let i in this.queue) {

counter++;

}

return counter;

}

}

C. Class Design Improvement with Design Patterns

Builder Pattern Implementation

interface IQueueBuilder {

setMaxSize(size: number): IQueueBuilder;

setType(type: string): IQueueBuilder;

build(): Queue;

}

class QueueBuilder implements IQueueBuilder {

private maxSize = 0;

private type = '';

setMaxSize(size: number): IQueueBuilder {

this.maxSize = size;

return this;

}

setType(type: string): IQueueBuilder {

this.type = type;

return this;

}

build(): Queue {

let queue: Queue;

switch (this.type) {

case 'fifo':

queue = new FIFOQueue();

break;

case 'lifo':

queue = new LIFOQueue();

break;

default:

// throw error

break;

}

queue.setMaxSize(this.maxSize);

return queue;

}

}

Decorator Pattern Implementation

abstract class QueueDecorator implements IQueueable {

decoratedQueue: IQueueable;

constructor(decoratedQueue: IQueueable) {

this.decoratedQueue = decoratedQueue;

}

enqueue(value: string): string[] {

this.decoratedQueue.enqueue(value);

return this.decoratedQueue.getQueue();

}

dequeue(): string {

return this.decoratedQueue.dequeue();

}

getQueue(): string[] {

return this.decoratedQueue.getQueue();

}

size(): number {

return this.decoratedQueue.size();

}

}

class MaxSizeQueueDecorator extends QueueDecorator {

private maxSize = 0;

setMaxSize(size: number): MaxSizeQueueDecorator {

this.maxSize = size;

return this;

}

enqueue(value: string): string[] {

if (this.decoratedQueue.size() < this.maxSize) {

this.decoratedQueue.enqueue(value);

} else {

console.log("Queue is full. Cannot enqueue.");

}

return this.decoratedQueue.getQueue();

}

}

Singleton Pattern Implementation

class SingletonQueue {

private static instance: SingletonQueue;

private queue: string[] = [];

private constructor() {}

public static getInstance(): SingletonQueue {

if (!SingletonQueue.instance) {

SingletonQueue.instance = new SingletonQueue();

}

return SingletonQueue.instance;

}

enqueue(value: string): string[] {

this.queue[this.queue.length] = value;

return this.queue;

}

dequeue(): string {

return this.queue.shift();

}

getQueue(): string[] {

return this.queue;

}

size(): number {

return this.queue.length;

}

}