COMP 2313 DATA STRUCTURES CODE ASSIGNMENT

|  |  |
| --- | --- |
| ***Assignment Name:*** *Stack and Queue* | ***Student Name :****Mustafa S Topsakal* |
| ***Assignment Date:*** *10/5/2020* |

# Problem

Implement Stack using any data structure (Array, ArrayList, LinkedList...)  Read Chapter 3!

* Implement 1 real life Stack problem example (Example:  **redo-undo** on any computer program)

Implement Double ended Queue using any data structure (Array, ArrayList, LinkedList...)  Read Chapter 3!

* Implement 1 real life Queue problem example (Example : In a Bank, Customer numerator)

.

# Challenges

Implement a stack file with array

Implement a dequeuer file with array

# Code

import java.util.\*;  
  
class Stack {  
 private int arr[];  
 private int top;  
 private int size;  
  
 //Constructor   
 Stack(int n) {  
 arr = new int[n];  
 size = n;  
 top = -1;  
 }  
  
 //return the size of the stack  
 public int size() {  
 return top + 1;  
 }  
  
 // stack is empty or not  
 public Boolean isEmpty() {  
 return top == -1; // or return size() == 0;  
 }  
  
 //full or not  
 public Boolean isFull() {  
 return top == size - 1; // or return size() == capacity;  
 }  
  
 // add an element temp in the stack  
 public void push(int temp) {  
 if (isFull()) {  
 System.*out*.println("OverFlow\nProgram Terminated\n");  
 System.*exit*(1);  
 }  
  
 System.*out*.println("Cake loading...Number " + temp);  
 arr[++top] = temp;  
 }  
  
 // pop top element from the stack  
 public int pop() {  
 // check for stack underflow  
 if (isEmpty()) {  
 System.*out*.println("IT'S EMPTY!!!");  
  
 }  
  
 System.*out*.println(stop());  
  
 // decrease stack size by 1 and (optionally) return the popped element  
 return arr[top--];  
 }  
  
 // show top element in a stack  
 public int stop() {  
 if (!isEmpty())  
 return arr[top];  
 else  
 System.*exit*(1);  
  
 return -1;  
 }  
  
  
  
}

import java.util.Scanner;  
  
public class Main {  
 public static void main(String[] args) {  
 Stack stack = new Stack(5);  
 stack.push(1);  
 stack.push(2);  
 stack.push(3);  
 stack.push(4);  
 stack.push(5);  
 //It's a simple machine that show either cake on the front or you can pass the display cake and look at the  
 // other cakes, i f you want it, you can pop it and take the cake  
  
 System.*out*.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");  
 System.*out*.println("It you'd like to see the cake in the machine press 1 or");  
 System.*out*.println("It you'd like to take the cake in the machine press 2 or");  
 System.*out*.println("It you'd like to take see the next cake press 3");  
 System.*out*.print("It you'd like to exit from machine press 4 :");  
 Scanner obj = new Scanner(System.*in*);  
 int num = obj.nextInt();  
 while(num != 4 && num< 5 && num>0){  
 if(num == 1){  
 System.*out*.println("Cake is "+stack.stop());  
 System.*out*.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");  
 }  
 else if(num == 2){  
  
 System.*out*.println("Cake out!"+ stack.pop());  
 System.*out*.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");  
 }  
 else if(num == 3){  
 stack.pop();  
 System.*out*.println("Cake is" + stack.stop());  
 System.*out*.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");  
 }  
  
 System.*out*.println("It you'd like to see the cake in the machine press 1 or");  
 System.*out*.println("It you'd like to take the cake in the machine press 2 or");  
 System.*out*.println("It you'd like to take see the next cake press 3");  
 System.*out*.print("It you'd like to exit from machine press 4 :");  
  
 num = obj.nextInt();  
 }  
  
  
 }  
}

public class DeQueue {  
 static final int *MAX* = 100;  
 int arr[];  
 int front,rear,size;  
  
 // constructor  
 public DeQueue(int n){  
 arr= new int[n];  
 front = -1;  
 rear = 0;  
 size = n;  
 }  
 //Dequeue full or not //  
 boolean isFull()  
 {  
 return ((front == 0 && rear == size-1)|| front == rear+1);  
 }  
  
 // Dequeue is empty or not //  
 boolean isEmpty ()  
 {  
 return (front == -1);  
 }  
  
 public void addFront(int temp){  
 if(isFull()){  
 System.*out*.println("Queue is full!");  
 return;  
 }  
 //if the first part is empty which is totally empty//  
 if (front == -1){  
 rear= 0;  
 front = 0;  
 }  
 //if the front shows the first data //  
 else if(front == 0){  
 front = size-1;  
  
 }  
 //decrement the front becasue we will add new data to the front //  
 else{  
 front= front-1;  
 arr[front] = temp;  
 }  
  
 }  
  
 public void addRear(int temp){  
 if (isFull())  
 {  
 System.*out*.println("Queue is full! ");  
 return;  
 }  
  
 // if the queue is totally empty//  
 if (front == -1)  
 {  
 rear = 0;  
 front = 0;  
  
 }  
  
 // if the last position is rear //  
 else if (rear == size-1)  
 rear = 0;  
  
 // increment the rea bc we will add new data to the end //  
 else{  
 rear = rear+1;  
 arr[rear] = temp ;  
 }  
  
  
  
  
 }  
 //display the front  
 public int stopFront(){  
 if (isEmpty())  
 {  
 return -1;  
 }  
 return arr[front];  
 }  
 //display the rear  
 public int stopRear(){  
  
 if(isEmpty() || rear < 0)  
 {  
  
 return -1;  
 }  
 return arr[rear];  
 }  
  
 public void deleteFront(){  
  
 if (isEmpty())  
 {  
 System.*out*.println("It's empty");  
 return ;  
 }  
  
 // if there is one element in the queue //  
 if (front == rear)  
 {  
 rear = -1;  
 front = -1;  
 ;  
 }  
 else  
 // delete the front and decrement the size also increment the front because we delete the previous one//  
 if (front == size -1)  
 front = 0;  
  
 else  
 front = front+1;  
 }  
  
 public void deleteEnd(){  
 if (isEmpty())  
 {  
 System.*out*.println("It's empty!");  
 return ;  
 }  
  
 // If there is one element  
 if (front == rear)  
 {  
 rear = -1;  
 front = -1;  
  
 }  
 //decrement the size by one  
 else if (rear == 0)  
 rear = size-1;  
 //decrement the rear position because we delete from bottom //  
 else  
 rear = rear-1;  
 }  
  
 public int getSize(){  
 return size;  
 }  
  
  
  
  
}

public class DeQueue {  
 static final int *MAX* = 100;  
 int arr[];  
 int front,rear,size;  
  
 // constructor  
 public DeQueue(int n){  
 arr= new int[n];  
 front = -1;  
 rear = 0;  
 size = n;  
 }  
 //Dequeue full or not //  
 boolean isFull()  
 {  
 return ((front == 0 && rear == size-1)|| front == rear+1);  
 }  
  
 // Dequeue is empty or not //  
 boolean isEmpty ()  
 {  
 return (front == -1);  
 }  
  
 public void addFront(int temp){  
 if(isFull()){  
 System.*out*.println("Queue is full!");  
 return;  
 }  
 //if the first part is empty which is totally empty//  
 if (front == -1){  
 rear= 0;  
 front = 0;  
 }  
 //if the front shows the first data //  
 else if(front == 0){  
 front = size-1;  
  
 }  
 //decrement the front becasue we will add new data to the front //  
 else{  
 front= front-1;  
 arr[front] = temp;  
 }  
  
 }  
  
 public void addRear(int temp){  
 if (isFull())  
 {  
 System.*out*.println("Queue is full! ");  
 return;  
 }  
  
 // if the queue is totally empty//  
 if (front == -1)  
 {  
 rear = 0;  
 front = 0;  
  
 }  
  
 // if the last position is rear //  
 else if (rear == size-1)  
 rear = 0;  
  
 // increment the rea bc we will add new data to the end //  
 else{  
 rear = rear+1;  
 arr[rear] = temp ;  
 }  
  
  
  
  
 }  
 //display the front  
 public int stopFront(){  
 if (isEmpty())  
 {  
 return -1;  
 }  
 return arr[front];  
 }  
 //display the rear  
 public int stopRear(){  
  
 if(isEmpty() || rear < 0)  
 {  
  
 return -1;  
 }  
 return arr[rear];  
 }  
  
 public void deleteFront(){  
  
 if (isEmpty())  
 {  
 System.*out*.println("It's empty");  
 return ;  
 }  
  
 // if there is one element in the queue //  
 if (front == rear)  
 {  
 rear = -1;  
 front = -1;  
 ;  
 }  
 else  
 // delete the front and decrement the size also increment the front because we delete the previous one//  
 if (front == size -1)  
 front = 0;  
  
 else  
 front = front+1;  
 }  
  
 public void deleteEnd(){  
 if (isEmpty())  
 {  
 System.*out*.println("It's empty!");  
 return ;  
 }  
  
 // If there is one element  
 if (front == rear)  
 {  
 rear = -1;  
 front = -1;  
  
 }  
 //decrement the size by one  
 else if (rear == 0)  
 rear = size-1;  
 //decrement the rear position because we delete from bottom //  
 else  
 rear = rear-1;  
 }  
  
 public int getSize(){  
 return size;  
 }  
  
  
  
  
}

# Pseudo Code

Stack()

Initialize the variables

IsEmpty

If start is 0 it’s empty

IsFull

If start= size it’s full

Push ()

Increase the size++ and add inside the arr[]

Pop()

If it’s empty

Remove the top element

Stop()

If it’s not empty

Show the top element without removing

Dequeu()

Initialize the variables

IsFull()

If rear shows the total size

IsEmpty()

If the front and rear shows the initial index

AddFront()

If it’s empty front = 0

If front = 0

Front = size -1

Else

Decrement the front

Add new data to arr[]

AddRear()

If it’s full

Return

If it’s empty

Rear=front=0

Else

Increment the rear index

Add to the arr[]

StopFront()

If it’s not empty

Show the front data

StopRear()

If it’s not empty

Show the last data

# UML Diagram

|  |
| --- |
| Dequeue |
| +arr[] : int  + front, rear, size : int |
| +Dequeue()  +isFull() : Boolean  +isEmpty() : Boolean  +addFront() : void  +addRear(): void  +stopFront()” int  +stoRear(): int  +deleteFront(): void  +deleteRear(): void  +getSize():int |

|  |
| --- |
| Stack |
| -arr[]: int  -top,size : int |
| Stack()  Size:int  IsEmpty(): Boolean  +isFull() : Boolean  Push(int): void  Pop(): int  Stop(): int |

# Outputs



