import java.util.Scanner;

// Step 1: Define StackInterface

interface StackInterface {

void push(int element);

int pop();

int peek();

void displayStack();

}

// Step 2: Define QueueInterface

interface QueueInterface {

void enqueue(int element);

int dequeue();

void displayQueue();

}

// Step 3: Define class StackQueue that implements both interfaces

class StackQueue implements StackInterface, QueueInterface {

private int[] stack;

private int[] queue;

private int top;

private int front, rear;

private int size;

// Constructor to initialize stack and queue

public StackQueue(int size) {

this.size = size;

stack = new int[size];

queue = new int[size];

top = -1;

front = -1;

rear = -1;

}

// Stack methods implementation

@Override

public void push(int element) {

if (top == size - 1) {

System.out.println("Stack Overflow");

} else {

stack[++top] = element;

System.out.println("Pushed: " + element);

}

}

@Override

public int pop() {

if (top == -1) {

System.out.println("Stack Underflow");

return -1;

} else {

int popped = stack[top--];

System.out.println("Popped: " + popped);

return popped;

}

}

@Override

public int peek() {

if (top == -1) {

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

return -1;

} else {

System.out.println("Top element: " + stack[top]);

return stack[top];

}

}

@Override

public void displayStack() {

if (top == -1) {

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

} else {

System.out.print("Stack elements: ");

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

System.out.print(stack[i] + " ");

}

System.out.println();

}

}

// Queue methods implementation

@Override

public void enqueue(int element) {

if (rear == size - 1) {

System.out.println("Queue Overflow");

} else {

if (front == -1) {

front = 0;

}

queue[++rear] = element;

System.out.println("Enqueued: " + element);

}

}

@Override

public int dequeue() {

if (front == -1 || front > rear) {

System.out.println("Queue Underflow");

return -1;

} else {

int dequeued = queue[front++];

System.out.println("Dequeued: " + dequeued);

return dequeued;

}

}

@Override

public void displayQueue() {

if (front == -1 || front > rear) {

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

} else {

System.out.print("Queue elements: ");

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

System.out.print(queue[i] + " ");

}

System.out.println();

}

}

}

// Step 5: Define StackQueueTest class

public class StackQueueTest {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.println("Enter the size of the stack and queue:");

int size = sc.nextInt();

StackQueue sq = new StackQueue(size);

int choice;

do {

System.out.println("\nChoose an operation:");

System.out.println("1. Push (Stack)");

System.out.println("2. Pop (Stack)");

System.out.println("3. Peek (Stack)");

System.out.println("4. Display Stack");

System.out.println("5. Enqueue (Queue)");

System.out.println("6. Dequeue (Queue)");

System.out.println("7. Display Queue");

System.out.println("8. Exit");

choice = sc.nextInt();

switch (choice) {

case 1:

System.out.println("Enter element to push:");

int elementPush = sc.nextInt();

sq.push(elementPush);

break;

case 2:

sq.pop();

break;

case 3:

sq.peek();

break;

case 4:

sq.displayStack();

break;

case 5:

System.out.println("Enter element to enqueue:");

int elementEnqueue = sc.nextInt();

sq.enqueue(elementEnqueue);

break;

case 6:

sq.dequeue();

break;

case 7:

sq.displayQueue();

break;

case 8:

System.out.println("Exiting...");

break;

default:

System.out.println("Invalid choice! Please try again.");

}

} while (choice != 8);

sc.close();

}

}