**Problem Statement 1: Working with Stacks**

1. **Write a Program to implement Stack using Array.**

**package** day8assignment;

**import** java.util.Scanner;

**public** **class** Stackarray {

**private** **static** **final** **int** ***MAX*** = 10;

**private** String[] stack;

**private** **int** top;

**public** Stackarray() {

stack = **new** String[***MAX***];

top = -1;

}

**public** **boolean** isFull() {

**return** top == ***MAX*** - 1;

}

**public** **boolean** isEmpty() {

**return** top == -1;

}

**public** **void** push(String value) {

**if** (isFull()) {

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

} **else** {

stack[++top] = value;

}

}

**public** String pop() {

**if** (isEmpty()) {

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

**return** **null**;

} **else** {

**return** stack[top--];

}

}

**public** **void** displayStack() {

**if** (isEmpty()) {

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();

}

}

**public** **static** **void** main(String[] args) {

Scanner sc = **new** Scanner(System.***in***);

Stackarray stack = **new** Stackarray();

System.***out***.println("Enter 4 elements to push onto the stack:");

**for** (**int** i = 0; i < 4; i++) {

System.***out***.print("Enter element " + (i + 1) + ": ");

String element = sc.nextLine();

stack.push(element);

}

System.***out***.println("After Pushing 4 Elements:");

stack.displayStack();

stack.pop();

System.***out***.println("After a Pop:");

stack.displayStack();

}

}

**Output:**

Enter 4 elements to push onto the stack:

Enter element 1: hello

Enter element 2: world

Enter element 3: java

Enter element 4: programming

After Pushing 4 Elements:

Stack elements: hello world java programming

After a Pop:

Stack elements: hello world java

**2. Write a Program to implement Stack using Linked List.**

package day8assignment;

import java.util.Scanner;

class Node {

float data;

Node next;

public Node(float data) {

this.data = data;

this.next = null;

}

}

public class Stacklinkedlist {

private Node top;

public Stacklinkedlist () {

top = null;

}

public boolean isEmpty() {

return top == null;

}

public void push(float value) {

Node newNode = new Node(value);

newNode.next = top;

top = newNode;

}

public float pop() {

if (isEmpty()) {

System.*out*.println("empty atack");

return -1;

} else {

float poppedValue = top.data;

top = top.next;

return poppedValue;

}

}

public void displayStack() {

Node current = top;

while (current != null) {

System.*out*.print(current.data + " ");

current = current.next;

}

System.*out*.println("null");

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.*in*);

Stacklinkedlist stack = new Stacklinkedlist ();

System.*out*.println("Enter 4 elements (float values) to push onto the stack:");

for (int i = 0; i < 4; i++) {

System.*out*.print("Enter element " + (i + 1) + ": ");

float element = scanner.nextFloat();

stack.push(element);

}

System.*out*.println("\nThe elements of the stack are:");

stack.displayStack();

stack.pop();

stack.pop();

System.*out*.println("\nAfter popping twice:");

stack.displayStack();

}

}

**Output**:

Enter 4 elements (float values) to push onto the stack:

Enter element 1: 10

Enter element 2: 20

Enter element 3: 30

Enter element 4: 40

The elements of the stack are:

40.0 30.0 20.0 10.0 null

After popping twice:

20.0 10.0 null

**3. Write a Program to Reverse a String using Stack.**

**package** day8assignment;

**import** java.util.Stack;

**import** java.util.Scanner;

**public** **class** Stackstring {

**public** **static** **void** main(String[] args) {

Scanner scanner = **new** Scanner(System.***in***);

System.***out***.print("Enter a string to reverse: ");

String input = scanner.nextLine();

String reversedString = *reverseString*(input);

System.***out***.println("Reversed String: " + reversedString);

}

**public** **static** String reverseString(String input) {

Stack<Character> stack = **new** Stack<>();

**for** (**int** i = 0; i < input.length(); i++) {

stack.push(input.charAt(i));

}

StringBuilder reversed = **new** StringBuilder();

**while** (!stack.isEmpty()) {

reversed.append(stack.pop());

}

**return** reversed.toString();

}

}

**Output**:

Enter a string to reverse: javaquiz

Reversed String: ziuqavaj

**4. Write a Program to evaluate an Expression using Stacks.**

package day8assignment;

import java.util.Stack;

import java.util.Scanner;

public class Expressionstack {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.*in*);

System.*out*.print("Enter an expression: ");

String expression = scanner.nextLine();

try {

int result = *evaluateExpression*(expression);

System.*out*.println("The result of the expression \"" + expression + "\" is: " + result);

} catch (ArithmeticException e) {

System.*out*.println("Error: " + e.getMessage());

}

}

public static int evaluateExpression(String expression) {

Stack<Integer> stack = new Stack<>();

String[] tokens = expression.split(" ");

int currentNumber = 0;

char operator = '+';

for (int i = 0; i < tokens.length; i++) {

String token = tokens[i];

if (token.matches("\\d+")) {

currentNumber = Integer.*parseInt*(token);

}

if (token.matches("[+\\-\*/]") || i == tokens.length - 1) {

if (operator == '+') {

stack.push(currentNumber);

} else if (operator == '-') {

stack.push(-currentNumber);

} else if (operator == '\*') {

stack.push(stack.pop() \* currentNumber);

} else if (operator == '/') {

if (currentNumber == 0) {

throw new ArithmeticException("Division by zero is not allowed.");

}

stack.push(stack.pop() / currentNumber);

}

if (i < tokens.length - 1) {

operator = token.charAt(0);

}

}

}

int result = 0;

while (!stack.isEmpty()) {

result += stack.pop();

}

return result;

}

}

**Output:**

Enter an expression: 10 + 2 \* 6

The result of the expression "10 + 2 \* 6" is: 22

**5. Write a program to reverse a stack using recursion, without using any loop.**

**package** day8assignment;

**import** java.util.Stack;

**import** java.util.Scanner;

**public** **class** Reversestack {

**public** **static** **void** reverse(Stack<Integer> stack) {

**if** (stack.isEmpty()) {

**return**;

}

**int** top = stack.pop();

*reverse*(stack);

*insertAtBottom*(stack, top);

}

**public** **static** **void** insertAtBottom(Stack<Integer> stack, **int** element) {

**if** (stack.isEmpty()) {

stack.push(element);

**return**;

}

**int** top = stack.pop();

*insertAtBottom*(stack, element);

stack.push(top);

}

**public** **static** **void** printStack(Stack<Integer> stack) {

**for** (Integer element : stack) {

System.***out***.print(element + " ");

}

System.***out***.println();

}

**public** **static** **void** main(String[] args) {

Scanner sc= **new** Scanner(System.***in***);

Stack<Integer> stack = **new** Stack<>();

System.***out***.print("Enter values to push onto the stack: ");

String input = sc.nextLine();

String[] inputValues = input.split(" ");

**for** (String value : inputValues) {

stack.push(Integer.*parseInt*(value));

}

*reverse*(stack);

System.***out***.println("Reversed Stack:");

*printStack*(stack);

}

}

**Output:**

Enter values to push onto the stack: 1 2 3 4 5

Reversed Stack:

5 4 3 2 1

**6. Write a program to find the minimum element in a stack.**

package day8assignment;

import java.util.Stack;

import java.util.Scanner;

public class Minstack {

public static int findMin(Stack<Integer> stack) {

if (stack.isEmpty()) {

return Integer.*MAX\_VALUE*;

}

int top = stack.pop();

int min = *findMin*(stack);

if (top < min) {

min = top;

}

stack.push(top);

return min;

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.*in*);

Stack<Integer> stack = new Stack<>();

System.*out*.print("Enter values to push onto the stack: ");

String input = sc.nextLine();

String[] inputValues = input.split(" ");

for (String value : inputValues) {

stack.push(Integer.*parseInt*(value));

}

int minElement = *findMin*(stack);

System.*out*.println("Minimum element in the stack: " + minElement);

}

}

**Output:**

Enter values to push onto the stack: 16 15 29 18 35

Minimum element in the stack: 15

**7. Given a stack of integers, find whether the top element of the stack is an even number or not. Return true if the top element is an even number, else return false.**

package day8assignment;

import java.util.\*;

import java.util.Scanner;

public class Topevenstack {

public static void main(String[] args) {

Scanner sc = new Scanner(System.*in*);

Stack <Integer>stack=new Stack<>();

System.*out*.print("Enter values to push onto the stack: ");

int input = sc.nextInt();

stack.push(input);

int top = stack.peek();

if(top%2==0) {

System.*out*.println("true");

}else {

System.*out*.println("false");

}

}

}

**Output:**

Enter values to push onto the stack: 40 30 25 15 6

true

**Problem Statement 2: Working with Queues**

**8. Write a Program to implement Queue using Array.**

package day8assignment;

import java.util.Scanner;

class Queue {

int front, rear, size;

int capacity;

int[] queue;

public Queue(int capacity) {

this.capacity = capacity;

this.size = 0;

this.front = 0;

this.rear = capacity - 1;

this.queue = new int[capacity];

}

boolean isFull() {

return size == capacity;

}

boolean isEmpty() {

return size == 0;

}

void enqueue(int item) {

if (isFull()) {

System.*out*.println("Queue is full! Cannot enqueue.");

return;

}

rear = (rear + 1) % capacity;

queue[rear] = item;

size++;

}

int dequeue() {

if (isEmpty()) {

System.*out*.println("Queue is empty! Cannot dequeue.");

return Integer.*MIN\_VALUE*;

}

int item = queue[front];

front = (front + 1) % capacity;

size--;

return item;

}

void display() {

if (isEmpty()) {

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

return;

}

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

System.*out*.print(queue[(front + i) % capacity] + " ");

}

System.*out*.println();

}

}

public class Queues {

public static void main(String[] args) {

Scanner sc = new Scanner(System.*in*);

System.*out*.print("Enter the capacity of the queue: ");

int capacity = sc.nextInt();

Queue q = new Queue(capacity);

System.*out*.print("Enter the number of elements to enqueue: ");

int numElements = sc.nextInt();

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

System.*out*.print("Enter element " + (i + 1) + ": ");

int element = sc.nextInt();

q.enqueue(element);

}

System.*out*.print("Elements in queue: ");

q.display();

q.dequeue();

System.*out*.print("After removing the first element: ");

q.display();

}

}

**Output:**

Enter the capacity of the queue: 5

Enter the number of elements to enqueue: 5

Enter element 1: 10

Enter element 2: 20

Enter element 3: 30

Enter element 4: 40

Enter element 5: 50

Elements in queue: 10 20 30 40 50

After removing the first element: 20 30 40 50

**9. Write a Program to implement Queue using Linked List.**

package day6assignment;

import java.util.Scanner;

class Node {

int data;

Node next;

public Node(int data) {

this.data = data;

this.next = null;

}

}

class Queue1 {

Node front, rear;

public Queue1() {

this.front = this.rear = null;

}

void enqueue(int item) {

Node newNode = new Node(item);

if (rear == null) {

front = rear = newNode;

return;

}

rear.next = newNode;

rear = newNode;

}

int dequeue() {

if (front == null) {

System.*out*.println("Queue is empty! Cannot dequeue.");

return Integer.*MIN\_VALUE*;

}

int item = front.data;

front = front.next;

if (front == null) {

rear = null;

}

return item;

}

void display() {

if (front == null) {

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

return;

}

Node temp = front;

while (temp != null) {

System.*out*.print(temp.data + " ");

temp = temp.next;

}

System.*out*.println();

}

}

public class Queuelinkedlist {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.*in*);

Queue1 q = new Queue1();

System.*out*.print("Enter the number of elements to enqueue: ");

int n = scanner.nextInt();

System.*out*.println("Enter " + n + " elements:");

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

q.enqueue(scanner.nextInt());

}

System.*out*.print("Elements in queue: ");

q.display();

System.*out*.print("Enter the number of elements to dequeue: ");

int m = scanner.nextInt();

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

q.dequeue();

}

System.*out*.print("After removing " + m + " elements: ");

q.display();

scanner.close();

}

}

**Output:**

Enter the number of elements to enqueue: 5

Enter 5 elements:

89 99 109 209 309

Elements in queue: 89 99 109 209 309

Enter the number of elements to dequeue: 1

After removing 1 elements: 99 109 209 309

**10. Write a Program to Implement Circular Queue using Array.**

**package** day8assignment;

**class** CircularQueue1 {

**int** front, rear, size;

**int** capacity;

**int**[] queue;

**public** CircularQueue1(**int** capacity) {

**this**.capacity = capacity;

**this**.size = 0;

**this**.front = 0;

**this**.rear = capacity - 1;

**this**.queue = **new** **int**[capacity];

}

**boolean** isFull() {

**return** size == capacity;

}

**boolean** isEmpty() {

**return** size == 0;

}

**void** enqueue(**int** item) {

**if** (isFull()) {

System.***out***.println("Queue is full! Cannot enqueue.");

**return**;

}

rear = (rear + 1) % capacity;

queue[rear] = item;

size++;

}

**int** dequeue() {

**if** (isEmpty()) {

System.***out***.println("Queue is empty! Cannot dequeue.");

**return** Integer.***MIN\_VALUE***;

}

**int** item = queue[front];

front = (front + 1) % capacity;

size--;

**return** item;

}

**void** display() {

**if** (isEmpty()) {

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

**return**;

}

**for** (**int** i = 0; i < size; i++) {

System.***out***.print(queue[(front + i) % capacity] + " ");

}

System.***out***.println();

}

}

**public** **class** Circularqueue {

**public** **static** **void** main(String[] args) {

CircularQueue1 q = **new** CircularQueue1(4);

q.enqueue(14);

q.enqueue(13);

q.enqueue(22);

q.enqueue(-8);

System.***out***.print("Elements in circular queue: ");

q.display();

q.dequeue();

System.***out***.print("After removing first element: ");

q.display();

}

}

**Output:**

Elements in circular queue: 14 13 22 -8

After removing first element: 13 22 -8

**11. Write a program to check whether a queue is empty or not.**

**package** day8assignment;

**import** java.util.Queue;

**import** java.util.LinkedList;

**public** **class** Emptyqueue {

**public** **static** **void** main(String args[]) {

Queue<String>queue = **new** LinkedList<>();

queue.add("Yellow");

queue.add("Green");

queue.add("Pink");

queue.add("Black");

queue.add("Blue");

queue.add("White");

**if**(queue.isEmpty()) {

System.***out***.println("queue is empty");

}**else** {

System.***out***.println("queue is not empty");

}

}

}

**Output:**

queue is not empty

**12. Given a queue, split the queue into two queues, one containing odd numbers and the other even numbers. The relative order of elements must be maintained in both the queues. Return an array containing the two queues, the 0th index should contain the queue of odd numbers and the 1st index should contain the queue of even numbers.**

package day8assignment;

import java.util.Queue;

import java.util.LinkedList;

public class Evenoroddqueue {

public static void main(String[] args) {

Queue<Integer> queue = new LinkedList<>();

Queue<Integer> evenQueue = new LinkedList<>();

Queue<Integer> oddQueue = new LinkedList<>();

queue.add(2);

queue.add(7);

queue.add(9);

queue.add(4);

queue.add(6);

queue.add(5);

queue.add(10);

for (int ele : queue) {

if (ele % 2 == 0) {

evenQueue.offer(ele);

} else {

oddQueue.offer(ele);

}

}

System.*out*.println("Even Queue: " + evenQueue);

System.*out*.println("Odd Queue: " + oddQueue);

}

}

**Output:**

Even Queue: [2, 4, 6, 10]

Odd Queue: [7, 9, 5]