***DAY 8-ASSIGNMENT***

***PRACTICE EXERCISE-8***

**//1. Write a Program to Implement Stack using Array****.**

package sowji;

class Stack {

private static final int *MAX\_SIZE* = 100;

private int[] arr;

private int top;

public Stack() {

arr = new int[*MAX\_SIZE*];

top = -1;

}

public void push(double d) {

if (top == *MAX\_SIZE* - 1) {

System.*out*.println("Stack Overflow: Cannot push " + d + ", stack is full.");

} else {

arr[++top] = (int) d;

System.*out*.println("Pushed " + d + " onto the stack.");

}

}

public int pop() {

if (top == -1) {

System.*out*.println("Stack Underflow: Cannot pop, stack is empty.");

return -1;

} else {

int poppedValue = arr[top--];

System.*out*.println("Popped " + poppedValue + " from the stack.");

return poppedValue;

}

}

public int peek() {

if (top == -1) {

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

return -1;

} else {

return arr[top];

}

}

public void display() {

}

public boolean isEmpty() {

return false;

}

}

public class Main1 {

public static void main(String[] args) {

Stack stack = new Stack();

 stack.push(4);

 stack.push(3);

 stack.push(2);

 stack.push(1);

 int poppedValue = stack.pop();

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

 int topElement = stack.peek();

 System.*out*.println("Top element of the stack: " + topElement);

 }

}

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

package sowji;

class Node {

   double data;

   Node next;

Node(double data) {

       this.data = data;

       this.next = null;

    }

}

class Stack {

   private Node top;

   public Stack() {

this.top = null;

   }

   public void push(double data) {

       Node newNode = new Node(data);

       newNode.next = top;

       top = newNode;

   }

   public double pop() {

       if (isEmpty()) {

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

           return -1;

       }

       double value = top.data;

       top = top.next;

       return value;

   }

   public boolean isEmpty() {

       return top == null;

   }

   public void display() {

       Node current = top;

       while (current != null) {

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

           current = current.next;

       }

       System.*out*.println("null");

   }

}

public class StackUsingLinkedList{

   public static void main(String[] args) {

   Stack stack = new Stack();

       stack.push(10.0);

       stack.push(20.0);

       stack.push(30.0);

       stack.push(40.0);

       System.*out*.print("The elements of the stack are: ");

       stack.display();

       stack.pop();

       System.*out*.print("After popping twice: ");

       stack.display();

    }

}

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

package sowji ;

import java.util.Stack;

public class ReverseStringUsingStack {

public static String reverseString(String input) {

int len = input.length();

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

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

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

}

StringBuilder reversed = new StringBuilder();

while (!stack.isEmpty()) {

reversed.append(stack.pop());

}

return reversed.toString();

}

public static void main(String[] args) {

String input = "Javaquiz";

String reversed = *reverseString*(input);

System.*out*.println("Input String: " + input);

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

}

}

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

package sowji;

import java.util.Stack;

public class ExpressionEvaluator {

public static int precedence(char op) {

if (op == '+' || op == '-') return 1;

if (op == '\*' || op == '/') return 2;

return 0;

}

public static int applyOp(int a, int b, char op) {

switch (op) {

case '+': return a + b;

case '-': return a - b;

case '\*': return a \* b;

case '/': return a / b;

}

return 0;

}

public static int evaluate(String expression) {

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

Stack<Character> ops = new Stack<>();

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

if (expression.charAt(i) == ' ') continue;

if (expression.charAt(i) >= '0' && expression.charAt(i) <= '9') {

StringBuilder sbuf = new StringBuilder();

while (i < expression.length() && expression.charAt(i) >= '0' && expression.charAt(i) <= '9')

sbuf.append(expression.charAt(i++));

values.push(Integer.*parseInt*(sbuf.toString()));

i--;

}

else {

while (!ops.isEmpty() && *precedence*(ops.peek()) >= *precedence*(expression.charAt(i))) {

int val2 = values.pop();

int val1 = values.pop();

char op = ops.pop();

values.push(*applyOp*(val1, val2, op));

}

ops.push(expression.charAt(i));

}

}

while (!ops.isEmpty()) {

int val2 = values.pop();

int val1 = values.pop();

char op = ops.pop();

values.push(*applyOp*(val1, val2, op));

}

return values.pop();

}

public static void main(String[] args) {

String expression = "10+2\*6";

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

}

}

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

package sowji ;

import java.util.Stack;

public class StackReverser {

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

if (stack.isEmpty()) {

stack.push(element);

return;

}

int topElement = stack.pop();

*insertAtBottom*(stack, element);

stack.push(topElement);

}

public static void reverseStack(Stack<Integer> stack) {

if (stack.isEmpty()) {

return;

}

int topElement = stack.pop();

*reverseStack*(stack);

*insertAtBottom*(stack, topElement);

}

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

for (int element : stack) {

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

}

System.*out*.println();

}

public static void main(String[] args) {

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

stack.push(1);

stack.push(2);

stack.push(3);

stack.push(4);

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

*printStack*(stack);

*reverseStack*(stack);

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

*printStack*(stack);

}

}

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

package sowji ;

import java.util.Stack;

 public class MinStack {

 private Stack<Integer> stack;

 private Stack<Integer> minStack;

  public MinStack() {

 stack = new Stack<>();

minStack = new Stack<>();

 }

   public void push(int x) {

 stack.push(x);

 if (minStack.isEmpty() || x <= minStack.peek()) {

 minStack.push(x);

 }

 }

 public void pop() {

 if (!stack.isEmpty()) {

 int top = stack.pop();

 if (top == minStack.peek()) {

 minStack.pop();

 }

 }

 }

   public int top() {

 return stack.peek();

 }

public int getMin() {

 return minStack.peek();

 }

public static void main(String[] args) {

 MinStack minStack = new MinStack();

 minStack.push(16);

 minStack.push(15);

 minStack.push(29);

 minStack.push(19);

 minStack.push(18);

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

 }

}

**//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 sowji ;

import java.util.Stack;

 public class EvenCheckStack {

 public static boolean isTopElementEven(Stack<Integer> stack) {

 if (stack.isEmpty()) {

 throw new IllegalArgumentException("Stack is empty.");

 }

 return stack.peek() % 2 == 0;

 }

public static void main(String[] args) {

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

 stack.push(40);

  try {

 boolean isEven = *isTopElementEven*(stack);

 System.*out*.println("Is the top element even? " + isEven);

 } catch (IllegalArgumentException e) {

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

 }

 }

}

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

package sowji;

public class Queue {

 private int front, rear, size;

 private int capacity;

 private int[] array;

 public Queue(int capacity) {

 this.capacity = capacity;

 front = this.size = 0;

 rear = capacity - 1;

 array = new int[this.capacity];

 }

 private boolean isFull() {

 return (this.size == this.capacity);

 }

 private boolean isEmpty() {

 return (this.size == 0);

 }

 public void enqueue(int item) {

 if (isFull()) {

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

 return;

 }

 this.rear = (this.rear + 1) % this.capacity;

 this.array[this.rear] = item;

 this.size = this.size + 1;

 System.*out*.println(item + " enqueued to queue");

 }

 public int dequeue() {

 if (isEmpty()) {

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

 return Integer.*MIN\_VALUE*;

 }

 int item = this.array[this.front];

 this.front = (this.front + 1) % this.capacity;

 this.size = this.size - 1;

 return item;

 }

 public void displayQueue() {

 if (isEmpty()) {

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

 return;

 }

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

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

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

 }

 System.*out*.println();

 }

public static void main(String[] args) {

 Queue queue = new Queue(10);

 queue.enqueue(10);

 queue.enqueue(20);

 queue.enqueue(30);

 queue.enqueue(40);

 queue.displayQueue();

 queue.dequeue();

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

 queue.displayQueue() ;

 }

}

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

package sowji;

class Node1 {

int data;

Node next;

public Node1(int data) {

this.data = data;

this.next = null;

}

}

public class Queue10 {

 private Node front, rear;

 public Queue10() {

 this.front = this.rear = null;

 }

 public void enqueue(int item) {

 Node newNode = new Node(item);

 if (this.rear == null) {

 this.front = this.rear = newNode;

 return;

 }

 this.rear.next = newNode;

 this.rear = newNode;

 }

 public int dequeue() {

 if (this.front == null) {

 throw new IllegalStateException("Queue is empty. Cannot dequeue");

 }

 int item = (int) this.front.data;

 this.front = this.front.next;

 if (this.front == null) {

 this.rear = null;

 }

 return item;

 }

 public void displayQueue() {

 if (this.front == null) {

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

 return;

 }

 Node temp = this.front;

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

 while (temp != null) {

 System.*out*.print(temp.data + " ");

 temp = temp.next;

 }

 System.*out*.println();

 }

public static void main(String[] args) {

 Queue10 queue = new Queue10();

   queue.enqueue(89);

 queue.enqueue(99);

 queue.enqueue(109);

 queue.enqueue(209);

 queue.enqueue(309);

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

 queue.displayQueue();

 queue.dequeue();

 queue.dequeue();

 System.*out*.println("After removing two elements:");

 queue.displayQueue();

 }

}

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

package sowji;

 public class CircularQueue {

 private int front, rear, size;

 private int capacity;

 private int[] array;

 public CircularQueue(int capacity) {

 this.capacity = capacity;

 this.front = this.size = 0;

 this.rear = capacity - 1;

 this.array = new int[this.capacity];

 }

  public boolean isFull() {

 return (this.size == this.capacity);

 }

  public boolean isEmpty() {

 return (this.size == 0);

 }

 public void enqueue(int item) {

 if (isFull()) {

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

 return;

 }

 this.rear = (this.rear + 1) % this.capacity;

 this.array[this.rear] = item;

 this.size++;

 System.*out*.println(item + " enqueued to queue");

 }

 public int dequeue() {

 if (isEmpty()) {

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

 return Integer.*MIN\_VALUE*;

 }

 int item = this.array[this.front];

 this.front = (this.front + 1) % this.capacity;

 this.size--;

 return item;

 }

 public void displayQueue() {

 if (isEmpty()) {

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

 return;

 }

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

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

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

 }

 System.*out*.println();

 }

public static void main(String[] args) {

 CircularQueue queue = new CircularQueue(5);

 queue.enqueue(14);

 queue.enqueue(13);

 queue.enqueue(22);

 queue.enqueue(-8);

 queue.displayQueue();

  queue.dequeue();

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

 queue.displayQueue();

 }

}

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

package sowji;

import java.util.LinkedList;

import java.util.Queue;

 public class QueueCheck {

 public static void main(String[] args) {

Queue<String> queue1 = new LinkedList<>();

Queue<String> queue2 = new LinkedList<>();

queue1.add("Yellow");

queue1.add("Green");

queue1.add("Pink");

queue1.add("Black");

queue1.add("Blue");

queue1.add("White");

if (queue1.isEmpty()) {

System.*out*.println("Queue 1: Empty");

} else {

System.*out*.println("Queue 1: Not Empty");

}

if (queue2.isEmpty()) {

System.*out*.println("Queue 2: Empty");

} else {

System.*out*.println("Queue 2: 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 sowji;

import java.util.LinkedList;

import java.util.Queue;

 public class SplitQueue {

 public static Queue<Integer>[] splitQueue(Queue<Integer> inputQueue) {

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

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

 while (!inputQueue.isEmpty()) {

 int number = inputQueue.poll();

 if (number % 2 == 0) {

 evenQueue.add(number);

 } else {

 oddQueue.add(number);

 }

 }

 return new Queue[]{oddQueue, evenQueue};

 }

 public static void printQueue(String message, Queue<Integer> queue) {

 System.*out*.print(message + ": ");

 while (!queue.isEmpty()) {

 System.*out*.print(queue.poll());

 }

 System.*out*.println();

 }

public static void main(String[] args) {

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

 inputQueue.add(2);

 inputQueue.add(7);

 inputQueue.add(9);

 inputQueue.add(4);

 inputQueue.add(6);

 inputQueue.add(5);

 inputQueue.add(1);

 inputQueue.add(0);

 Queue<Integer>[] resultQueues = *splitQueue*(inputQueue);

 Queue<Integer> oddQueue = resultQueues[0];

 Queue<Integer> evenQueue = resultQueues[1];

*printQueue*("Odd Queue", oddQueue);

*printQueue*("Even Queue", evenQueue);

}

}