DAY-8 PRACTICE EXERCISE

BASICS OF JAVA

1. ArrayStack1

**package** Testing2;

**import** java.util.Scanner;

**public** **class** ArrayStack1 {

**private** String[] arrStack;

**private** **int** index;

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

**public** ArrayStack1(**int** size) {

arrStack = **new** String[size];

index = -1;

top = 0;

}

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

**if** (index == arrStack.length - 1) {

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

} **else** {

arrStack[++index] = value;

top++;

}

}

**public** **void** isEmpty() {

System.***out***.println(index == -1);

}

**public** **void** isFull() {

System.***out***.println(index == arrStack.length - 1);

}

**private** **void** display() {

**if** (index == -1) {

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

} **else** {

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

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

}

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

}

}

**private** **void** pop() {

**if** (index == -1) {

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

} **else** {

System.***out***.println("Popped Element: " + arrStack[index]);

arrStack[index--] = **null**;

top--;

}

}

**private** **void** peak() {

**if** (index == -1) {

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

} **else** {

System.***out***.println("Top Element: " + arrStack[index]);

}

}

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

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

**int** size=sc.nextInt();

ArrayStack1 as = **new** ArrayStack1(size);

as.push("Hello");

as.push("World");

as.push("Java");

as.push("Programming");

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

as.display();

as.isFull();

as.isEmpty();

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

as.pop();

as.display();

as.peak();

}

}

2. LinkedStack Class

**package** Testing2;

**import** java.util.Scanner;

**class** Node {

**double** data;

Node next;

**public** Node(**double** data) {

**this**.data = data;

**this**.next = **null**;

}

}

**public** **class** LinkedStack {

Node top;

**public** LinkedStack() {

**this**.top = **null**;

}

**private** **void** push(**double** d) {

Node node = **new** Node(d);

node.next = top;

top = node;

}

**private** **void** display() {

**if** (top == **null**) {

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

} **else** {

Node current = top;

**while** (current != **null**) {

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

current = current.next;

}

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

}

}

**private** **double** pop() {

**if** (top == **null**) {

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

**return** -1;

} **else** {

**double** val= top.data;

top=top.next;

**return** val;

}

}

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

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

LinkedStack stack = **new** LinkedStack();

System.***out***.println("How many You Want To Enter : ");

**int** num=sc.nextInt();

System.***out***.println("Enter Elements : ");

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

**double** data=sc.nextDouble();

stack.push(data);

}

System.***out***.print("Elements are : ");

stack.display();

stack.pop();

stack.pop();

System.***out***.print("After Pop Elements : ");

stack.display();

}

}

**3.** ReverseStringUsingStack

**package** Testing2;

**import** java.util.Scanner;

**import** java.util.Stack;

**public** **class** ReverseStringUsingStack {

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

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

**for** (**char** c : input.toCharArray())

stack.push(c);

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

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

reversed.append(stack.pop());

}

**return** reversed.toString();

}

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

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

System.***out***.println("Enter String : ");

String input=sc.next();//JavaQuiz

String reversed = *reverseString*(input);

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

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

}

}

4. ExpressionSolveByStack

**package** Testing2;

**import** java.util.Scanner;

**import** java.util.Stack;

**public** class ExpressionSolveByStack {

**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** (Character.isDigit(expression.charAt(i))) {

StringBuilder buffer = new StringBuilder();

**while** (i < expression.length() && Character.isDigit(expression.charAt(i)))

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

values.push(Integer.parseInt(buffer.toString()));

i--;

} **else if** (expression.charAt(i) == '+' || expression.charAt(i) == '-' || expression.charAt(i) == '\*'

|| expression.charAt(i) == '/') {

**while** (!ops.isEmpty() && hasPrecedence(expression.charAt(i), ops.peek()))

values.push(applyOp(ops.pop(), values.pop(), values.pop()));

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

}

}

**while** (!ops.isEmpty())

values.push(applyOp(ops.pop(), values.pop(), values.pop()));

**return** values.pop();

}

**public static** boolean hasPrecedence(char op1, char op2) {

**if** (op2 == '(' || op2 == ')')

return false;

**if** ((op1 == '\*' || op1 == '/') && (op2 == '+' || op2 == '-'))

return false;

**else**

return true;

}

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

switch (op) {

case '+':

return a + b;

case '-':

return a - b;

case '\*':

return a \* b;

case '/':

if (b == 0)

throw new UnsupportedOperationException("Cannot divide by zero");

return a / b;

}

return 0;

}

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

Scanner sc = new Scanner(System.in);

**System**.out.println("Enter Expression :");

String exp = sc.next();// "10+2\*6"

**System**.out.println("Output : " + evaluate(exp)); // Output: 22

}

}

5. StackReverseOperation

**package** Testing2;

**import** java.util.Stack;

**public** **class** StackReverseOperation {

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

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

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

*reverseOpration*(stack);

*insertBottom*(stack, temp);

}

}

**private** **static** **void** insertBottom(Stack<Integer> stack, **int** value) {

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

stack.push(value);

} **else** {

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

*insertBottom*(stack, value);

stack.push(temp);

}

}

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

**for** (Integer 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:");

*displayStackElements*(stack);

*reverseOpration*(stack);

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

*displayStackElements*(stack);

}

}

6. FindMinElementInStack Class

**package** Testing2;

**import** java.util.Stack;

**public** **class** FindMinElementInStack {

**public** **static** **int** isMinimum(Stack<Integer> stack) {

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

**throw** **new** IllegalArgumentException("Stack is Empty");

}

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

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

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

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

**if** (current < min) {

min = current;

}

tempStack.push(current);

}

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

stack.push(tempStack.pop());

}

**return** min;

}

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

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

stack.push(16);

stack.push(15);

stack.push(29);

stack.push(19);

stack.push(18);

**try** {

**int** minElement = *isMinimum*(stack);

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

} **catch** (IllegalArgumentException e) {

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

}

}

}

7. StackTopEvenOrNot Class

**package** Testing2;

**import** java.util.Stack;

**public** **class** StackTopEvenOrNot {

**public** **static** **boolean** isTopEven(Stack<Integer> stack) {

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

**throw** **new** IllegalStateException("Stack is empty");

}

**int** topElement = stack.peek();

**return** topElement % 2 == 0;

}

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

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

stack.push(40);

stack.push(30);

stack.push(25);

stack.push(15);

**boolean** result = *isTopEven*(stack);

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

}

}

8. ArrayQueue Class

**package** Testing2;

**public** **class** ArrayQueue {

**private** **int**[] queue;

**private** **int** front,rear,capacity,size;

**public** ArrayQueue(**int** capacity) {

**this**.capacity = capacity;

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

front = 0;

rear = -1;

size = 0;

}

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

**if** (isFull()) {

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

**return**;

}

rear = (rear + 1) % capacity;

queue[rear] = item;

size++;

}

**public** **int** dequeue() {

**if** (isEmpty()) {

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

**return** -1;

}

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

front = (front + 1) % capacity;

size--;

**return** item;

}

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

**return** size == 0;

}

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

**return** size == capacity;

}

**public** **void** display() {

**if** (isEmpty()) {

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

**return**;

}

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

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

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

}

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

}

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

ArrayQueue q = **new** ArrayQueue(5);

q.enqueue(10);

q.enqueue(20);

q.enqueue(30);

q.enqueue(40);

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

q.display();

q.dequeue();

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

q.display();

}

}

9. LinkedQueue Class

**package** Testing2;

**import** java.util.Scanner;

**class** Node {

**int** data;

Node next;

**public** Node(**int** data) {

**this**.data = data;

**this**.next = **null**;

}

}

**public** **class** LinkedQueue {

**private** Node front;

**private** Node rear;

**private** **int** size;

**public** LinkedQueue() {

**this**.front = **null**;

**this**.rear = **null**;

**this**.size = 0;

}

**private** **void** enqueue(**int** data) {

Node newNode = **new** Node(data);

**if** (rear == **null**) {

front = rear = newNode;

} **else** {

rear.next = newNode;

rear = newNode;

}

size++;

}

**private** **int** dequeue() {

**if** (isEmpty()) {

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

**return** -1;

} **else** {

**int** value = front.data;

front = front.next;

**if** (front == **null**) {

rear = **null**;

}

size--;

**return** value;

}

}

**private** **boolean** isEmpty() {

**return** front == **null**;

}

**private** **int** getSize() {

**return** size;

}

**private** **void** display() {

**if** (isEmpty()) {

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

} **else** {

Node current = front;

**while** (current != **null**) {

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

current = current.next;

}

}

}

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

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

LinkedQueue queue = **new** LinkedQueue();

System.***out***.println("How many elements do you want to enter?");

**int** num = sc.nextInt();

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

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

**int** data = sc.nextInt();

queue.enqueue(data);

}

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

queue.display();

queue.dequeue();

queue.dequeue();

System.***out***.print("\nAfter removing two elements: ");

queue.display();

}

}

10. CircularQueue

**package** Testing2;

**public** **class** CircularQueue {

**private** **int**[] queue;

**private** **int** front, rear, capacity, size;

**public** CircularQueue(**int** capacity) {

**this**.capacity = capacity;

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

front = -1;

rear = -1;

size = 0;

}

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

**if** (isFull()) {

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

**return**;

}

rear = (rear + 1) % capacity;

queue[rear] = item;

**if** (front == -1) {

front = rear;

}

size++;

}

**public** **int** dequeue() {

**if** (isEmpty()) {

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

**return** -1;

}

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

front = (front + 1) % capacity;

size--;

**if** (size == 0) {

front = -1;

rear = -1;

}

**return** item;

}

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

**return** size == 0;

}

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

**return** size == capacity;

}

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

CircularQueue q = **new** CircularQueue(5);

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 the first element : ");

q.display();

}

}

11.

**package** Testing2;

**import** java.util.LinkedList;

**import** java.util.Queue;

**public** class QueueEmptyOrNot {

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

// Input 1: Queue is Not Empty

Queue<String> queue1 = 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("Empty");

} **else** {

**System**.out.println("Not Empty");

}

// Input 2: Queue is Empty

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

**if** (queue2.isEmpty()) {

System.out.println("Empty");

} **else** {

System.out.println("Not Empty");

}

}

}

12. SplitQueue Class

**package** Testing2;

**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.offer(number);

} **else** {

oddQueue.offer(number);

}

}

**return** new Queue[] { oddQueue, evenQueue };

}

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

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

inputQueue.offer(2);

inputQueue.offer(7);

inputQueue.offer(9);

inputQueue.offer(4);

inputQueue.offer(6);

inputQueue.offer(5);

inputQueue.offer(10);

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

**System**.out.println("Odd: " + result[0]);

**System**.out.println("Even: " + result[1]);

}

}