**Day-8 Basics of Java**

Problem Statement 1: Working with Stacks

1. //stack class

**package** com.mycom.stack.array;

**public** **class** Stack {

**private** String[] stack;

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

**private** **int** capacity;

**public** Stack(**int** capacity) {

**this**.capacity = capacity;

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

top = -1;

}

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

**if** (top == capacity - 1) {

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

**return**;

}

stack[++top] = element;

}

**public** String pop() {

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

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

**return** **null**;

}

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

}

**public** **void** printStack() {

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

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

}

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

}

}

//main class

**public** **class** MainStack {

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

// **TODO** Auto-generated method stub

Stack stack = **new** Stack(10);

stack.push("Hello");

stack.push("world");

stack.push("java");

stack.push("Programming");

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

stack.printStack();

// Popping one element

stack.pop();

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

stack.printStack();

}

}

2. // stack class

**package** com.mycom.stack.linkedlist;

**class** Stack {

**double** data;

Stack next;

**public** Stack(**double** data) {

**this**.data = data;

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

}

}

**class** Stack1 {

**private** Stack top;

**public** Stack1() {

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

}

**public** **void** push(**double** data) {

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

newNode.next = top;

top = newNode;

}

**public** **double** pop() {

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

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

**return** -1;

}

**double** poppedData = top.data;

top = top.next;

**return** poppedData;

}

**public** **void** printStack() {

Stack currentNode = top;

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

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

currentNode = currentNode.next;

}

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

}

}

// Main class

**public** **class** MainStack {

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

Stack1 stack = **new** Stack1();

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.printStack();

stack.pop();

stack.pop();

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

stack.printStack();

}

}

3. **package** com.mycom.stack.reversestring;

**import** com.mycom.stack.array.Stack;

**public** **class** ReverseString {

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

String input = "Java Quiz";

String reversed = *reverseString*(input);

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

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

}

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

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

**for** (**char** c : str.toCharArray()) {

stack.push(c);

}

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

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

reversed.append(stack.pop());

}

**return** reversed.toString();

}

}

4. **package com.mycom.stack.expression**;

**import** java.util.Stack;

**public** **class** EvaluateExpressionUsingStacks{

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

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

**int** result = *evaluateExpression*(expression);

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

}

**public** **static** **int** evaluateExpression(String expression) {

expression = expression.replaceAll("\\s", "");

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

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

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

**char** c = expression.charAt(i);

**if** (Character.*isDigit*(c)) {

**int** num = 0;

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

num = num \* 10 + (expression.charAt(i) - '0');

i++;

}

i--;

numbers.push(num);

}

**else** **if** (c == '+' || c == '-' || c == '\*' || c == '/') {

**while** (!operators.isEmpty() && *precedence*(operators.peek()) >= *precedence*(c)) {

**int** b = numbers.pop();

**int** a = numbers.pop();

**char** op = operators.pop();

numbers.push(*applyOperator*(a, b, op));

}

operators.push(c);

}

}

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

**int** b = numbers.pop();

**int** a = numbers.pop();

**char** op = operators.pop();

numbers.push(*applyOperator*(a, b, op));

}

**return** numbers.pop();

}

**public** **static** **int** precedence(**char** op) {

**switch** (op) {

**case** '+':

**case** '-':

**return** 1;

**case** '\*':

**case** '/':

**return** 2;

}

**return** -1;

}

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

**switch** (op) {

**case** '+':

**return** a + b;

**case** '-':

**return** a - b;

**case** '\*':

**return** a \* b;

**case** '/':

**if** (b == 0) {

**throw** **new** ArithmeticException("Cannot divide by zero");

}

**return** a / b;

}

**return** 0;

}

}

5.**package** com.mycom.stack.recursion;

**import** java.util.Stack;

**public** **class** ReverseStack {

**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: " + stack);

*reverseStack*(stack);

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

}

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

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

**return**;

}

**int** bottom = *popBottom*(stack);

*reverseStack*(stack);

stack.push(bottom);

}

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

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

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

**return** top;

} **else** {

**int** bottom = *popBottom*(stack);

stack.push(top);

**return** bottom;

}

}

}

6. .**package** com.mycom.stack.element;

**import** java.util.Stack;

**public** **class** MinElementInStack {

**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);

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

System.***out***.println("Minimum Element: " + *getMin*(stack));

}

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

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

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

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

**if** (minStack.isEmpty() || element < minStack.peek()) {

minStack.push(element);

} **else** {

minStack.push(minStack.peek());

}

}

**return** minStack.peek();

}

}

**7. package** com.mycom.stack.EvenOdd;

**import** java.util.Stack;

**public** **class** TopEvenOdd {

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

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

stack.push(15);

stack.push(25);

stack.push(30);

stack.push(40);

System.***out***.println(*isTopElementEven*(stack)); // Output: true

}

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

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

**return** **false**;

}

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

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

}

}

8. **package** com.mycom.queue.array;

**public** **class** Queue {

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

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

**public** Queue(**int** size) {

front = rear = 0;

capacity = size;

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

}

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

**if** (capacity == rear) {

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

**return**;

} **else** {

queue[rear] = item;

rear++;

}

}

**public** **void** dequeue() {

**if** (front == rear) {

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

**return**;

} **else** {

**for** (**int** i = 0; i < rear - 1; i++) {

queue[i] = queue[i + 1];

}

**if** (rear < capacity)

queue[rear] = 0;

rear--;

}

}

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

**if** (front == rear) {

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

**return**;

}

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

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

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

}

9. **package** com.mycom.queue.linklist;

**class** Linkedlist {

**int** data;

Linkedlist next;

**public** Linkedlist(**int** data) {

**this**.data = data;

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

}

}

**class** Queue {

**private** Linkedlist front, rear;

**public** Queue() {

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

}

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

Linkedlist newNode = **new** Linkedlist(item);

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

**this**.front = **this**.rear = newNode;

**return**;

}

**this**.rear.next = newNode;

**this**.rear = newNode;

}

**public** **void** dequeue() {

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

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

**return**;

}

**this**.front = **this**.front.next;

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

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

}

}

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

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

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

**return**;

}

Linkedlist temp = 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) {

Queue q = **new** Queue();

// Enqueue elements

q.enqueue(89);

q.enqueue(99);

q.enqueue(109);

q.enqueue(209);

q.enqueue(309);

// Display elements in queue

q.display();

// Dequeue two elements

q.dequeue();

q.dequeue();

// Display elements in queue after removing two elements

q.display();

}

}

10.**package** com.mycom.queue.circularqueue;

**class** CircularQueue{

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

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

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

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

front = -1;

rear = -1;

size = capacity;

}

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

**if** ((rear + 1) % size == front) {

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

**return**;

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

front = 0;

rear = 0;

queue[rear] = item;

} **else** {

rear = (rear + 1) % size;

queue[rear] = item;

}

}

**public** **void** dequeue() {

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

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

**return**;

} **else** **if** (front == rear) {

front = -1;

rear = -1;

} **else** {

front = (front + 1) % size;

}

}

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

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

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

**return**;

}

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

**int** i = front;

**while** (**true**) {

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

**if** (i == rear) {

**break**;

}

i = (i + 1) % size;

}

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

}

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

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

// Enqueue elements

q.enqueue(14);

q.enqueue(13);

q.enqueue(22);

q.enqueue(-8);

// Display elements in queue

q.display();

// Dequeue an element

q.dequeue();

// Display elements in queue after removing the first element

q.display();

}

}

11. **package** com.mycom.queue.emptyornot;

**import** java.util.LinkedList;

**import** java.util.Queue;

**public** **class** QueueCheck {

**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");

*checkQueue*(queue);

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

*checkQueue*(queue1);

}

**public** **static** **void** checkQueue(Queue<String> queue) {

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

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

} **else** {

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

}

}

}

12. **package** com.mycom.queue.split;

**import** java.util.LinkedList;

**import** java.util.Queue;

**public** **class** SplitQueueEvenOdd {

**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(10);

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

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

**for** (**int** num : result[0]) {

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

}

System.***out***.print("\nEven Queue: ");

**for** (**int** num : result[1]) {

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

}

}

@SuppressWarnings("unchecked")

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

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

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

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

**int** num = inputQueue.poll();

**if** (num % 2 == 0) {

evenQueue.add(num);

} **else** {

oddQueue.add(num);

}

}

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

}

}