**Scenario Based Assignment – POOJYANTH M**

**1. Browser History (Using Stack)**

**Scenario:** Simulate a web browser’s back and forward functionality using two stacks.

**Features:**

* **Visit new page:** Push to back Stack, clear forward Stack.
* **Go back:** Pop from back Stack and push to forward Stack.
* **Go forward:** Pop from forward Stack and push to back Stack.

**CODE**

import java.util.Stack;

public class BrowserHistory {

private Stack<String> backStack;

private Stack<String> forwardStack;

public BrowserHistory(String homepage) {

backStack = new Stack<>();

forwardStack = new Stack<>();

backStack.push(homepage);

}

public void visit(String url) {

backStack.push(url);

forwardStack.clear();

}

public String back() {

if (backStack.size() > 1) {

forwardStack.push(backStack.pop());

}

return backStack.peek();

}

public String forward() {

if (!forwardStack.isEmpty()) {

backStack.push(forwardStack.pop());

}

return backStack.peek();

}

// For demonstration: print current status

public void printStatus() {

System.out.println("Back stack: " + backStack);

System.out.println("Forward stack: " + forwardStack);

System.out.println("Current page: " + backStack.peek());

System.out.println();

}

public static void main(String[] args) {

BrowserHistory bh = new BrowserHistory("Home");

bh.printStatus();

bh.visit("Page1");

bh.printStatus();

bh.visit("Page2");

bh.printStatus();

bh.back();

bh.printStatus();

bh.back();

bh.printStatus();

bh.forward();

bh.printStatus();

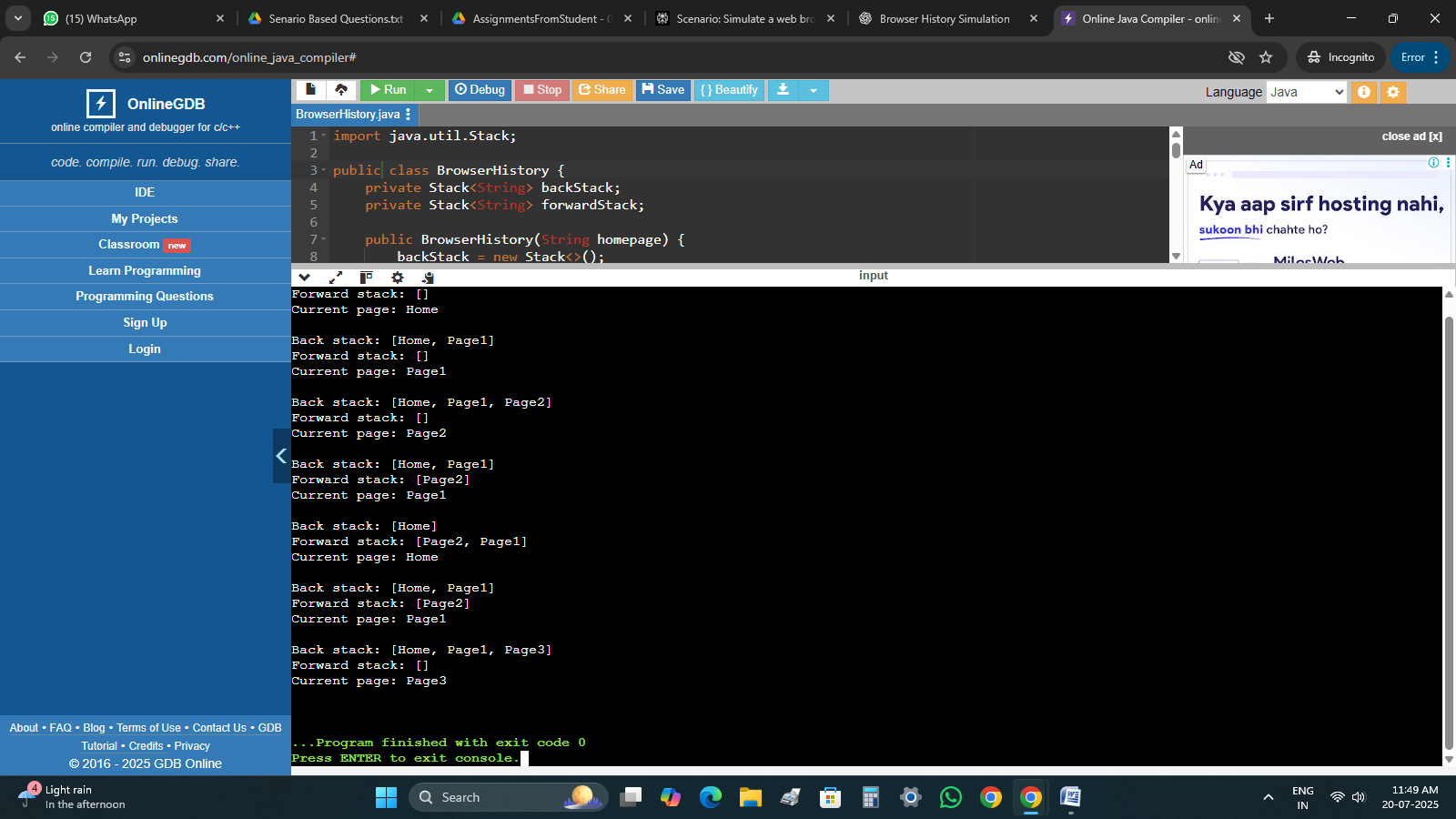
bh.visit("Page3");

bh.printStatus();

}

}

**OUTPUT**



**2. Print Queue (Using Linked List as Queue)**

**Scenario:** Simulate a printer that handles print jobs in FIFO order.

**Features:**

* Add new print jobs
* Process jobs in order
* View pending jobs

**CODE**

import java.util.\*;

class SpoolTower {

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

void queueJob(String fileName) {

docTrail.offer(fileName);

System.out.println("Job added: " + fileName);

}

void printJob() {

if (!docTrail.isEmpty()) {

System.out.println("Printing: " + docTrail.poll());

} else {

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

}

}

void viewQueue() {

System.out.println("Pending Jobs: " + docTrail);

}

public static void main(String[] args) {

SpoolTower printer = new SpoolTower();

printer.queueJob("Document1.pdf");

printer.queueJob("Image2.png");

printer.queueJob("Report3.docx");

printer.viewQueue();

printer.printJob();

printer.viewQueue();

printer.printJob();

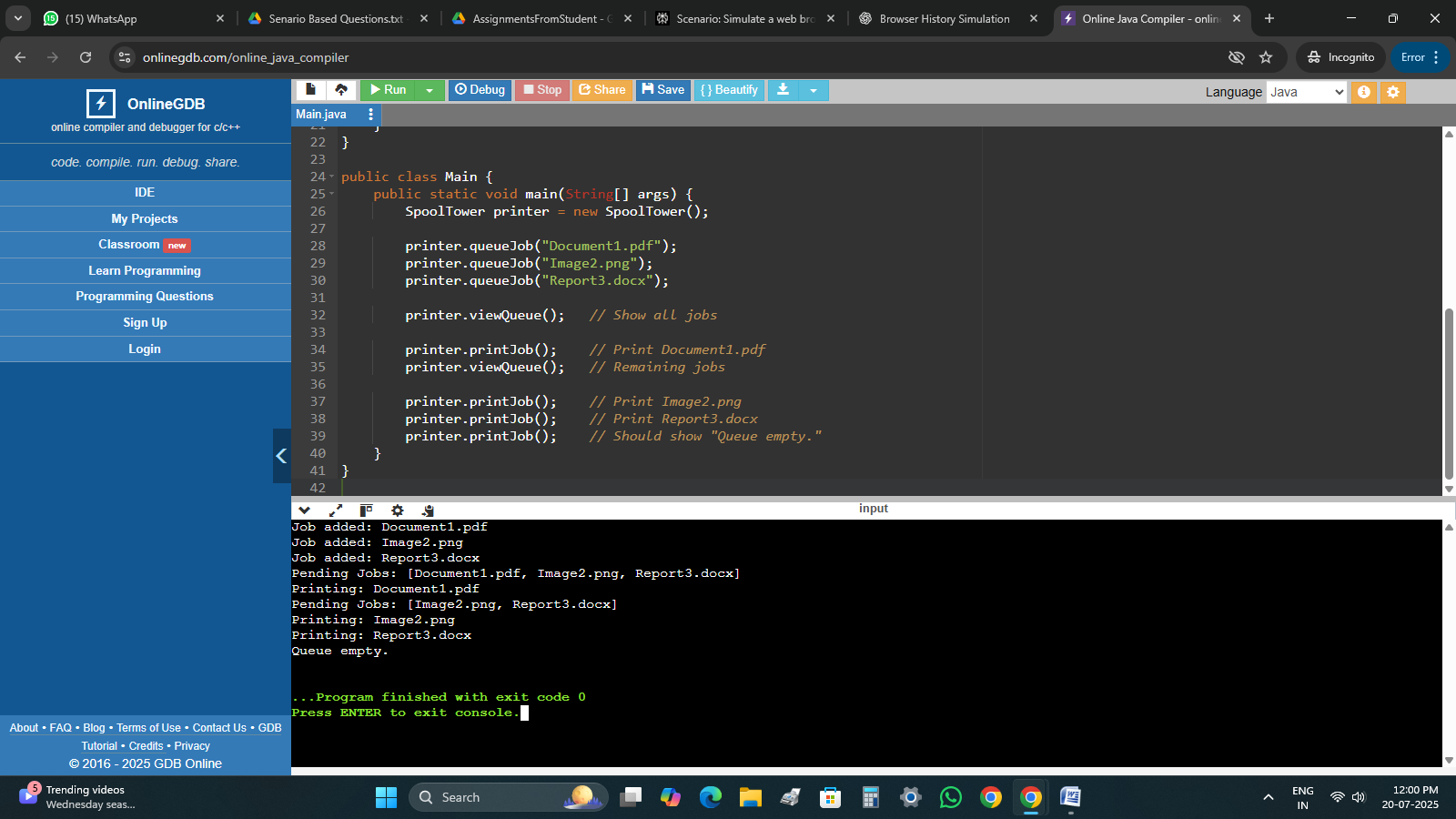
printer.printJob();

printer.printJob();

}

}

**OUTPUT**

****

**3. Hospital Bed Management (Using Linked List)**

**Scenario:** Track patients occupying hospital beds.

**Features:**

* Assign bed to new patient
* Discharge patient (remove by name or ID)
* Display current occupancy

**CODE**

import java.util.\*;

class MediWing {

LinkedList<String> bedTracker = new LinkedList<>();

void admitPatient(String patientID) {

bedTracker.add(patientID);

System.out.println("Admitted: " + patientID);

}

void releasePatient(String patientID) {

if (bedTracker.remove(patientID)) {

System.out.println("Discharged: " + patientID);

} else {

System.out.println("Patient not found.");

}

}

void displayBeds() {

System.out.println("Occupied Beds: " + bedTracker);

}

}

public class Main {

public static void main(String[] args) {

MediWing ward = new MediWing();

ward.admitPatient("P001");

ward.admitPatient("P002");

ward.admitPatient("P003");

ward.displayBeds();

ward.releasePatient("P002");

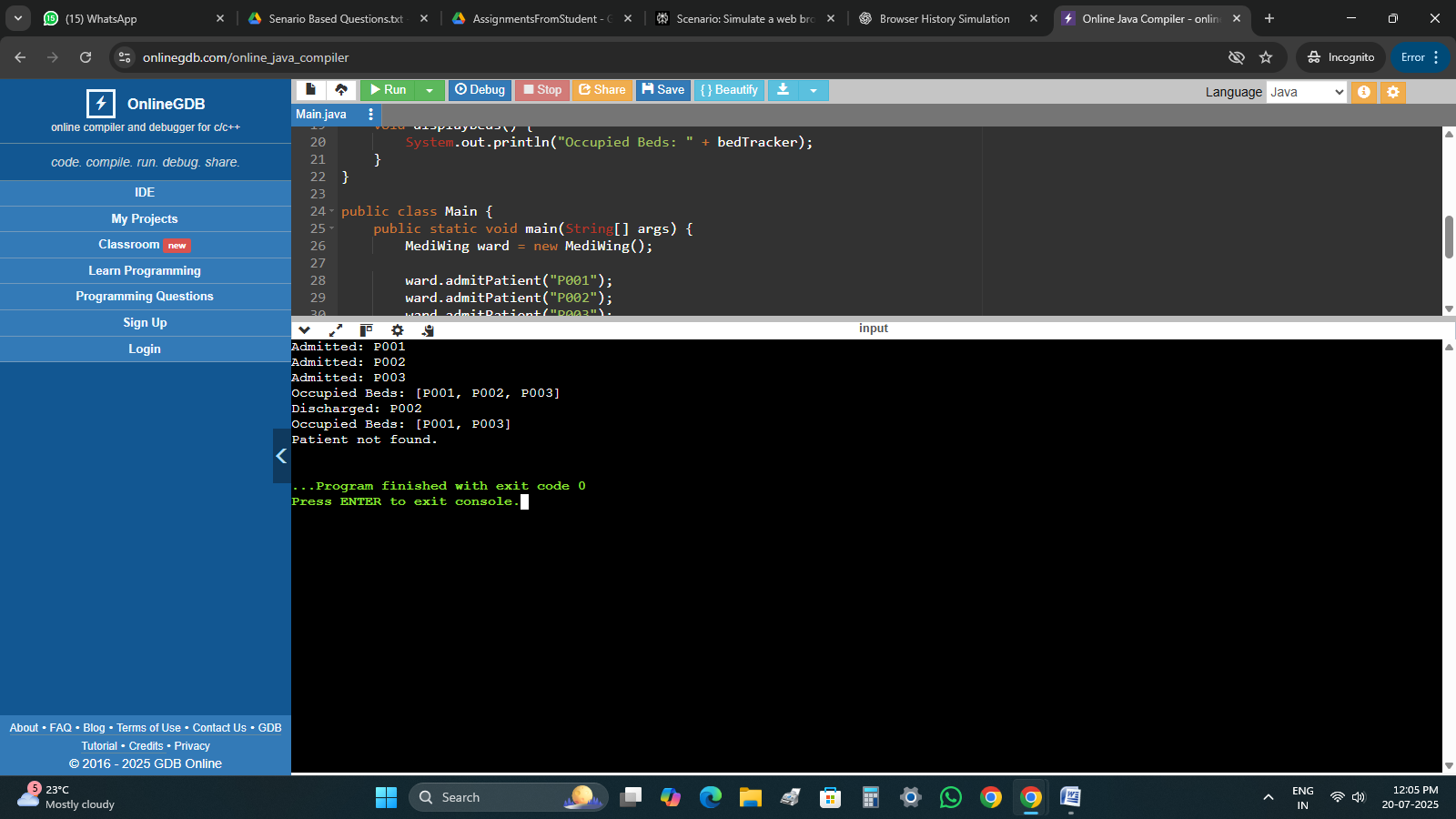
ward.displayBeds();

ward.releasePatient("P005");

}

}

**OUTPUT**



**4. Undo-Redo Function (Using Stack)**

**Scenario:** Track document edits with undo and redo.

**Features:**

* Perform an action → Push to undo Stack
* Undo → Move action to redo Stack
* Redo → Move back to undo Stack

**CODE**

import java.util.\*;

class DocManipulator {

Stack<String> pastActions = new Stack<>();

Stack<String> redoStack = new Stack<>();

void doAction(String act) {

pastActions.push(act);

redoStack.clear();

System.out.println("Action performed: " + act);

}

void undo() {

if (!pastActions.isEmpty()) {

String last = pastActions.pop();

redoStack.push(last);

System.out.println("Undo: " + last);

} else {

System.out.println("Nothing to undo.");

}

}

void redo() {

if (!redoStack.isEmpty()) {

String redo = redoStack.pop();

pastActions.push(redo);

System.out.println("Redo: " + redo);

} else {

System.out.println("Nothing to redo.");

}

}

}

public class Main {

public static void main(String[] args) {

DocManipulator editor = new DocManipulator();

editor.doAction("Type 'Hello'");

editor.doAction("Bold text");

editor.doAction("Insert image");

editor.undo();

editor.undo();

editor.redo();

editor.redo();

editor.undo();

editor.undo();

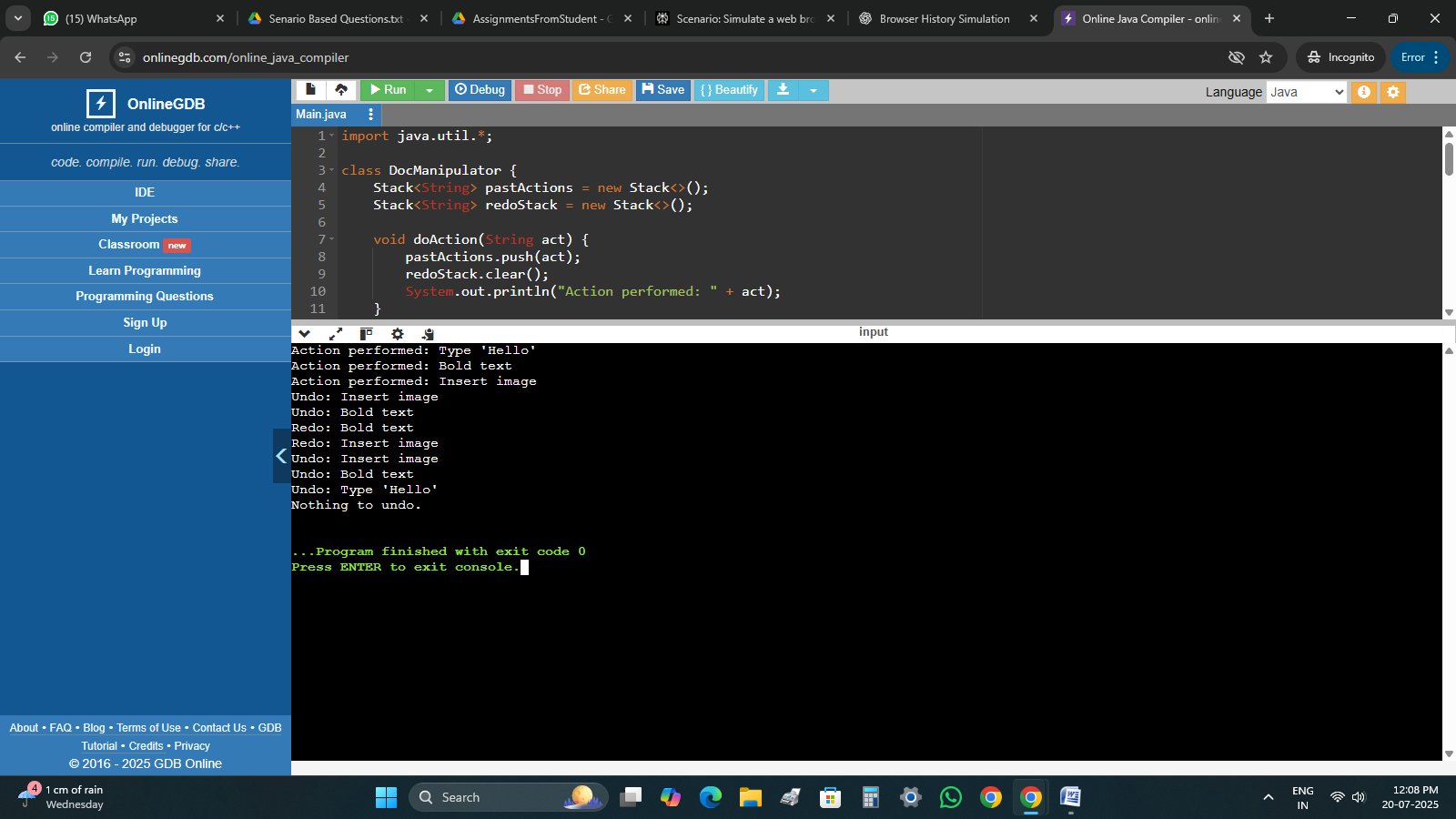
editor.undo();

editor.undo();

}

}

**OUTPUT**

****

**5. Ticket Booking System (Using Queue)**

**Scenario:**People are queued to book movie/train tickets.

**Features:**

* Add person to booking queue
* Serve next person (dequeue)
* Cancel ticket (remove specific person)

**CODE**

import java.util.\*;

class TicketSystem {

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

void bookTicket(String name) {

queue.offer(name);

System.out.println(name + " added to the booking queue.");

}

void serveTicket() {

if (!queue.isEmpty()) {

System.out.println("Ticket booked for: " + queue.poll());

} else {

System.out.println("No one in queue.");

}

}

void cancelTicket(String name) {

if (queue.remove(name)) {

System.out.println("Booking cancelled for: " + name);

} else {

System.out.println(name + " not found in queue.");

}

}

void showQueue() {

System.out.println("Current queue: " + queue);

}

}

public class Main {

public static void main(String[] args) {

TicketSystem system = new TicketSystem();

system.bookTicket("POOJYANTH");

system.bookTicket("RAKSHITHA");

system.bookTicket("DIVYASHRE");

system.showQueue();

system.serveTicket();

system.cancelTicket("RAKSHITHA");

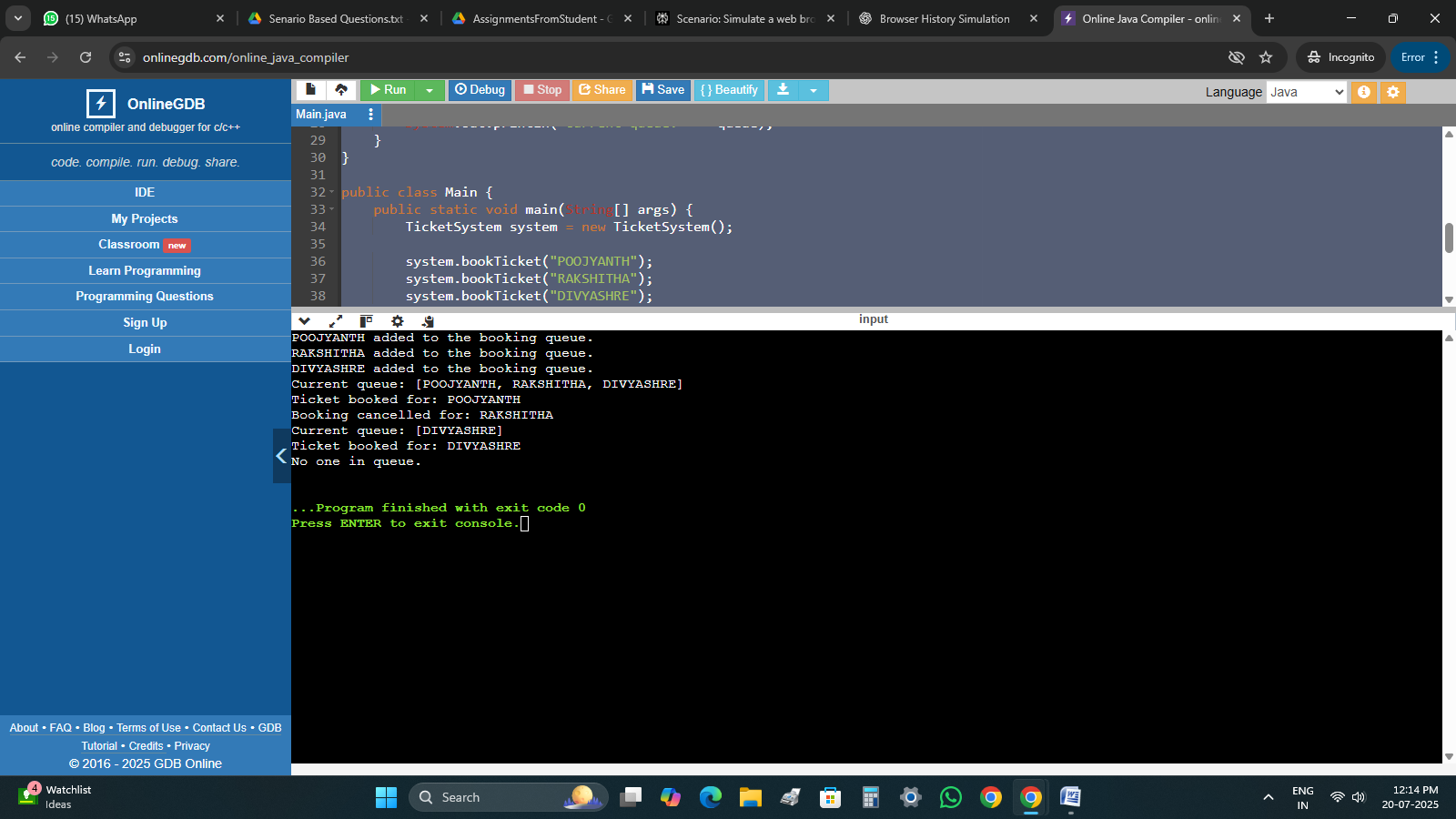
system.showQueue();

system.serveTicket();

system.serveTicket();

}

**OUTPUT**

****

**6. Car Wash Service Queue**

**Scenario:** Cars line up at a car wash center.

**Features:**

* Add normal cars to the end
* VIP cars go to the front
* Remove car after washing

**CODE**

import java.util.\*;

class CarWashQueue {

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

void addNormalCar(String carNumber) {

queue.addLast(carNumber);

System.out.println("Normal car added: " + carNumber);

}

void addVipCar(String carNumber) {

queue.addFirst(carNumber);

System.out.println("VIP car added: " + carNumber);

}

void washCar() {

if (!queue.isEmpty()) {

System.out.println("Washed car: " + queue.removeFirst());

} else {

System.out.println("No cars to wash.");

}

}

void showQueue() {

System.out.println("Waiting cars: " + queue);

}

}

public class Main {

public static void main(String[] args) {

CarWashQueue wash = new CarWashQueue();

wash.addNormalCar("KA05QJ6434");

wash.addNormalCar("KA13N7594");

wash.addVipCar("VIP2004");

wash.showQueue();

wash.washCar();

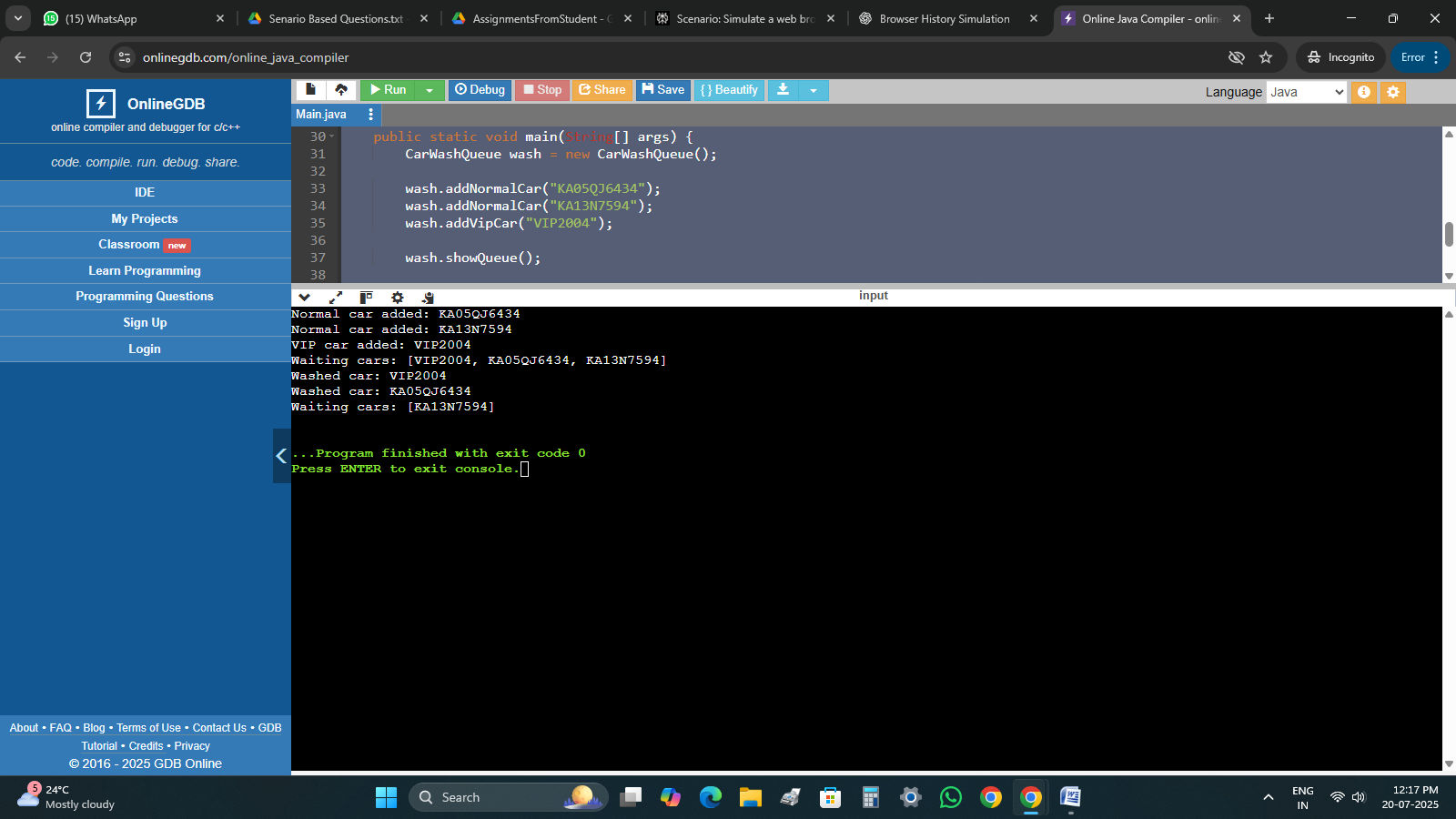
wash.washCar();

wash.showQueue();

}

}

**OUTPUT**

****

**7. Library Book Stack (Using Stack)**

**Scenario:**Books are stacked in a last-in-first-out order.

**Features:**

* Add book (push)
* Remove book (pop)
* Peek top book

**CODE**

import java.util.\*;

class LibraryStack {

Stack<String> bookStack = new Stack<>();

void addBook(String bookName) {

bookStack.push(bookName);

System.out.println("Book added: " + bookName);

}

void removeBook() {

if (!bookStack.isEmpty()) {

System.out.println("Book removed: " + bookStack.pop());

} else {

System.out.println("No books to remove.");

}

}

void peekTopBook() {

if (!bookStack.isEmpty()) {

System.out.println("Top book: " + bookStack.peek());

} else {

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

}

}

void showStack() {

System.out.println("Current Stack: " + bookStack);

}

}

public class Main {

public static void main(String[] args) {

LibraryStack library = new LibraryStack();

library.addBook("Squid Game");

library.addBook("Alice in Borderland");

library.addBook("50 shades of India");

library.showStack();

library.peekTopBook();

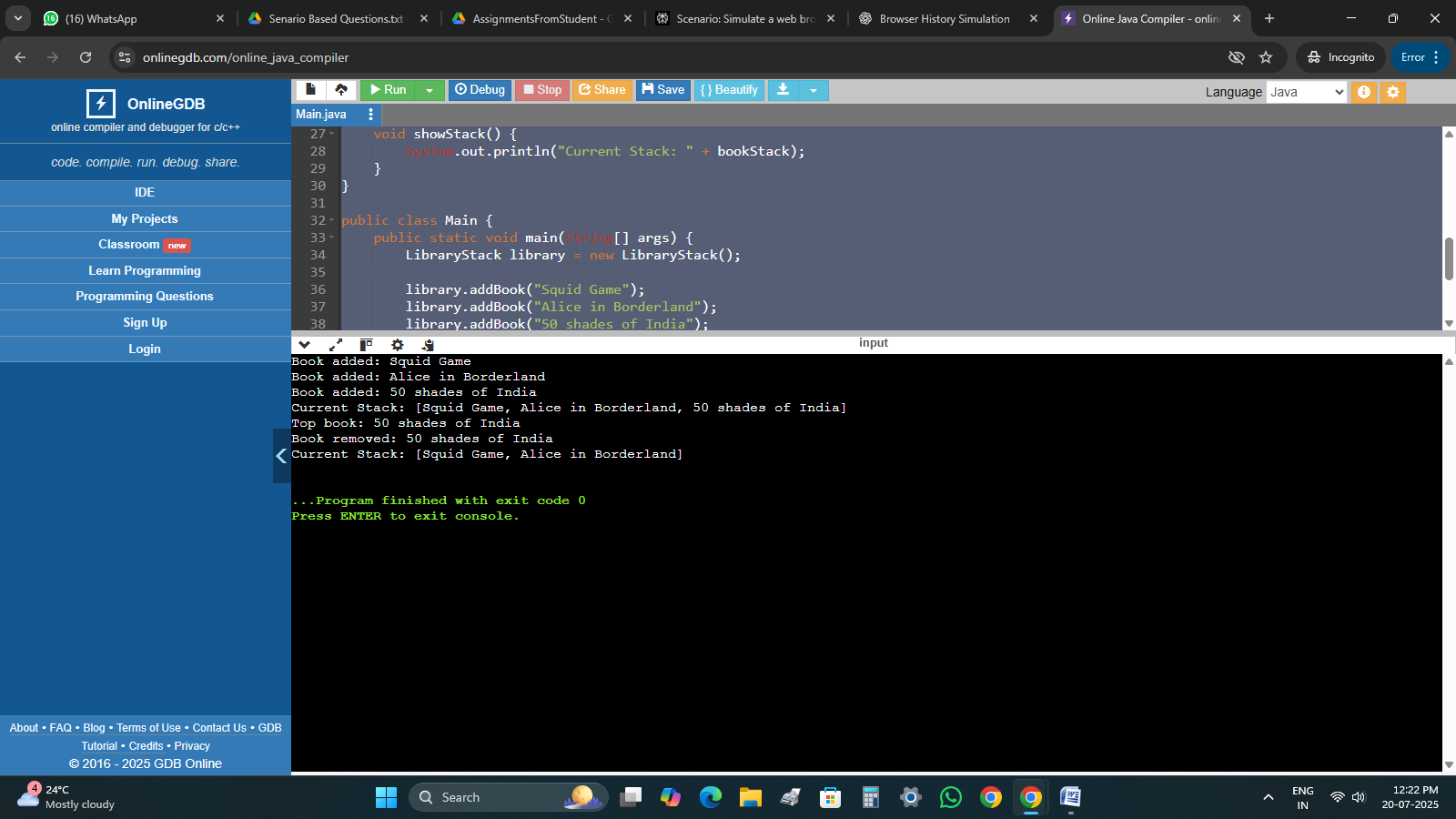
library.removeBook();

library.showStack();

}

}

**OUTPUT**

****

**8. Expression Evaluator (Infix to Postfix & Evaluate)**

**Scenario:**Create a calculator to evaluate expressions.

**Features:**

* Convert infix to postfix
* Evaluate postfix using stack

**CODE**

import java.util.\*;

public class Main {

static int precedence(char op) {

switch(op) {

case '+': case '-': return 1;

case '\*': case '/': return 2;

case '^': return 3;

default: return 0;

}

}

static boolean isOperator(char c) {

return c == '+' || c == '-' || c == '\*' || c == '/' || c == '^';

}

static String infixToPostfix(String infix) {

StringBuilder output = new StringBuilder();

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

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

char c = infix.charAt(i);

if (Character.isWhitespace(c)) continue;

if (Character.isDigit(c)) {

while (i < infix.length() && (Character.isDigit(infix.charAt(i)) || infix.charAt(i) == '.')) {

output.append(infix.charAt(i));

i++;

}

output.append(' ');

i--;

} else if (c == '(') {

stack.push(c);

} else if (c == ')') {

while (!stack.isEmpty() && stack.peek() != '(') {

output.append(stack.pop()).append(' ');

}

stack.pop();

} else if (isOperator(c)) {

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

output.append(stack.pop()).append(' ');

}

stack.push(c);

}

}

while (!stack.isEmpty()) {

output.append(stack.pop()).append(' ');

}

return output.toString().trim();

}

static double evaluatePostfix(String postfix) {

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

String[] tokens = postfix.split("\\s+");

for (String token : tokens) {

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

stack.push(Double.parseDouble(token));

} else if (token.length() == 1 && isOperator(token.charAt(0))) {

double b = stack.pop();

double a = stack.pop();

switch(token.charAt(0)) {

case '+': stack.push(a + b); break;

case '-': stack.push(a - b); break;

case '\*': stack.push(a \* b); break;

case '/': stack.push(a / b); break;

case '^': stack.push(Math.pow(a, b)); break;

}

}

}

return stack.pop();

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.println("Enter infix expression:");

String infix = sc.nextLine();

String postfix = infixToPostfix(infix);

System.out.println("Postfix: " + postfix);

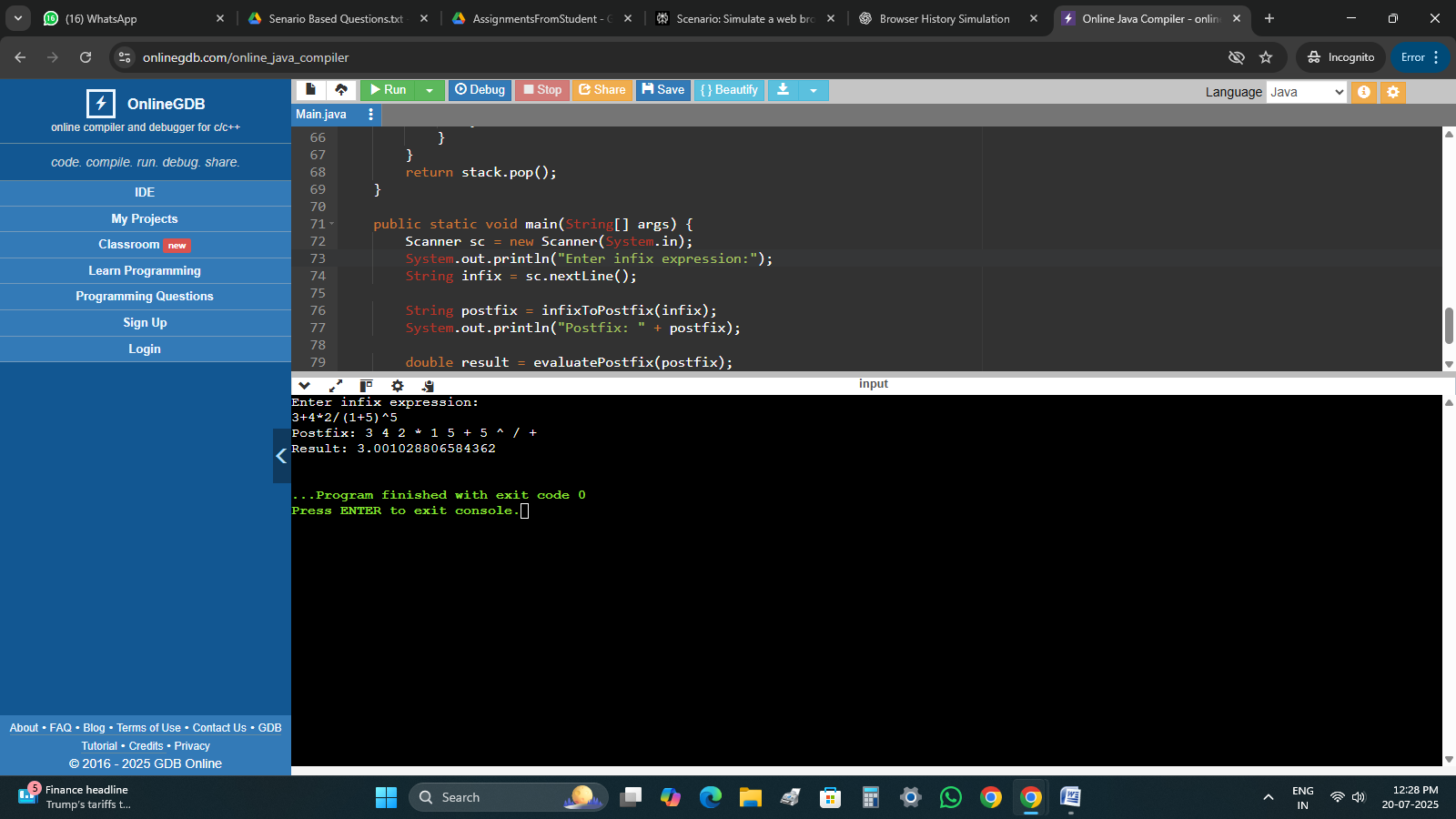
double result = evaluatePostfix(postfix);

System.out.println("Result: " + result);

}

}

**OUTPUT**

****

**9. Reverse Queue Using Stack**

**Scenario:** Reverse the order of a customer service queue.

**Features:**

* Enqueue customers
* Reverse using stack
* Display new order

**CODE**

import java.util.\*;

public class Main {

public static void main(String[] args) {

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

Scanner sc = new Scanner(System.in);

System.out.println("Enter number of customers:");

int n = sc.nextInt();

sc.nextLine();

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

System.out.print("Enter customer name: ");

queue.offer(sc.nextLine());

}

System.out.println("Original Queue: " + queue);

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

while (!queue.isEmpty()) {

stack.push(queue.poll());

}

while (!stack.isEmpty()) {

queue.offer(stack.pop());

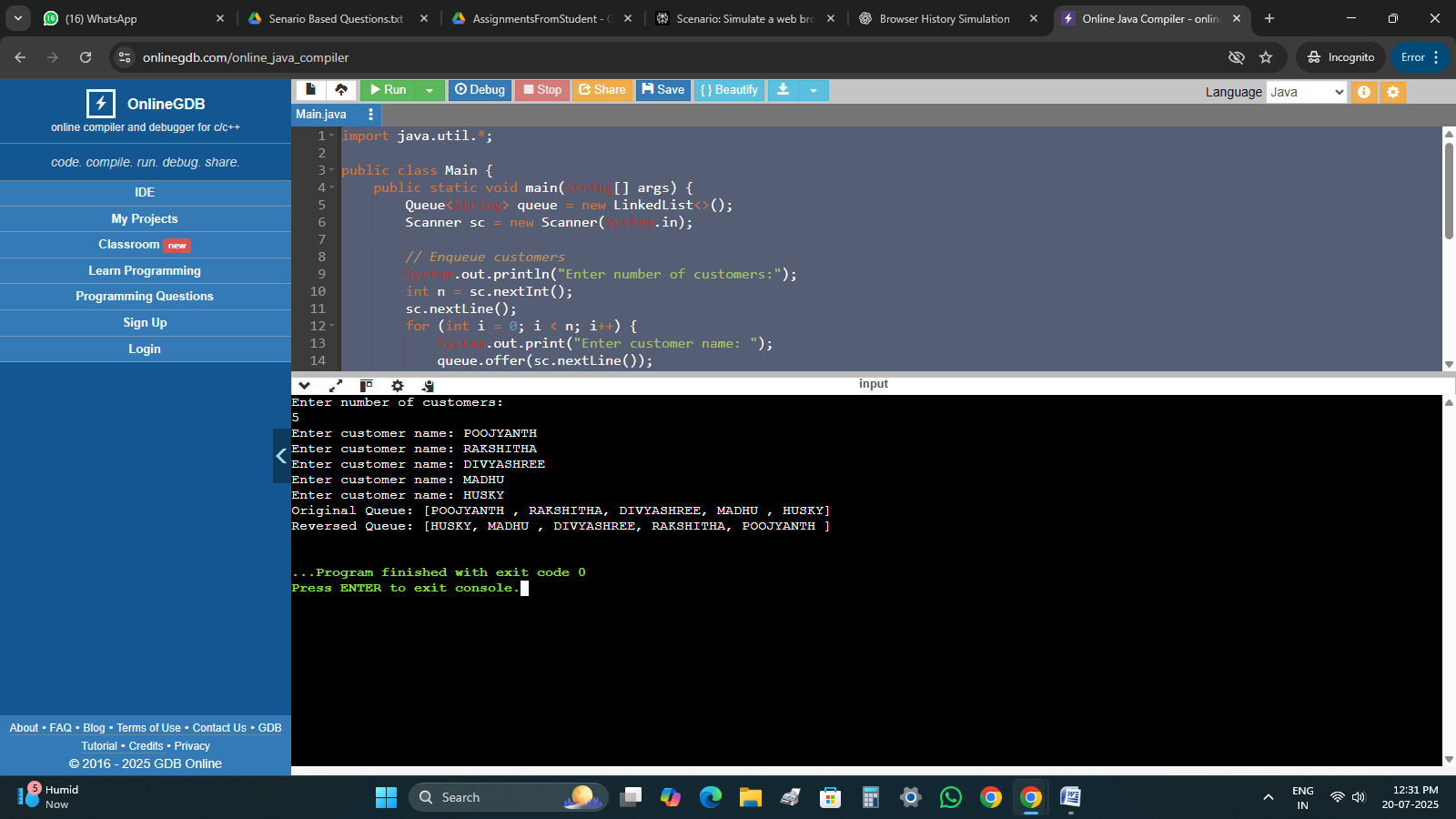
}

System.out.println("Reversed Queue: " + queue);

}

}

**OUTPUT**

****

**10. Student Admission Queue with Emergency Slot**

**Scenario:** College admission line where VIP quota students are handled first.

**Features:**

* Add student normally (end)
* Add VIP (front)
* Remove admitted student

**CODE**

import java.util.\*;

public class Main {

public static void main(String[] args) {

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

Scanner sc = new Scanner(System.in);

while (true) {

System.out.println("\n--- Student Admission Queue ---");

System.out.println("1. Add Normal Student");

System.out.println("2. Add VIP Student");

System.out.println("3. Admit Student (Remove Front)");

System.out.println("4. Show Queue");

System.out.println("5. Exit");

System.out.print("Choose an option: ");

int choice = sc.nextInt();

sc.nextLine();

switch (choice) {

case 1:

System.out.print("Enter student name: ");

String normal = sc.nextLine();

queue.addLast(normal);

break;

case 2:

System.out.print("Enter VIP student name: ");

String vip = sc.nextLine();

queue.addFirst(vip);

break;

case 3:

if (!queue.isEmpty()) {

System.out.println("Admitted: " + queue.removeFirst());

} else {

System.out.println("No students in queue.");

}

break;

case 4:

System.out.println("Current Queue: " + queue);

break;

case 5:

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

return;

default:

System.out.println("Invalid choice.");

}

}

}

}

**OUTPUT**

