Maintain a patient queue using a linked list where

* Patients can be added (registered) at the end.
* Emergency patients should be added at the beginning.
* Support removing discharged patients.

import java.util.\*;

public class Main {

    static class Patient {

        String name;

        Patient next;

        public Patient(String name) {

            this.name = name;

            this.next = null;

        }

    }

    static class PatientQueue {

        private Patient head;

        public void addPatient(String name) {

            Patient newPatient = new Patient(name);

            if (head == null) {

                head = newPatient;

            } else {

                Patient temp = head;

                while (temp.next != null) {

                    temp = temp.next;

                }

                temp.next = newPatient;

            }

        }

        public void addEmergencyPatient(String name) {

            Patient newPatient = new Patient(name);

            newPatient.next = head;

            head = newPatient;

        }

        public void dischargePatient(String name) {

            if (head == null) {

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

                return;

            }

            if (head.name.equals(name)) {

                head = head.next;

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

                return;

            }

            Patient prev = head;

            Patient curr = head.next;

            while (curr != null && !curr.name.equals(name)) {

                prev = curr;

                curr = curr.next;

            }

            if (curr == null) {

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

            } else {

                prev.next = curr.next;

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

            }

        }

        public void displayQueue() {

            if (head == null) {

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

                return;

            }

            Patient temp = head;

            System.out.print("Patient Queue: ");

            while (temp != null) {

                System.out.print(temp.name + " -> ");

                temp = temp.next;

            }

            System.out.println("null");

        }

    }

    public static void main(String[] args) {

        PatientQueue queue = new PatientQueue();

        queue.addPatient("John");

        queue.addPatient("Emma");

        queue.addEmergencyPatient("Drake");

        queue.addPatient("Sophia");

        queue.addEmergencyPatient("Mia");

        System.out.println("Initial Patient Queue:");

        queue.displayQueue();

        System.out.println("\nDischarging patient: Emma");

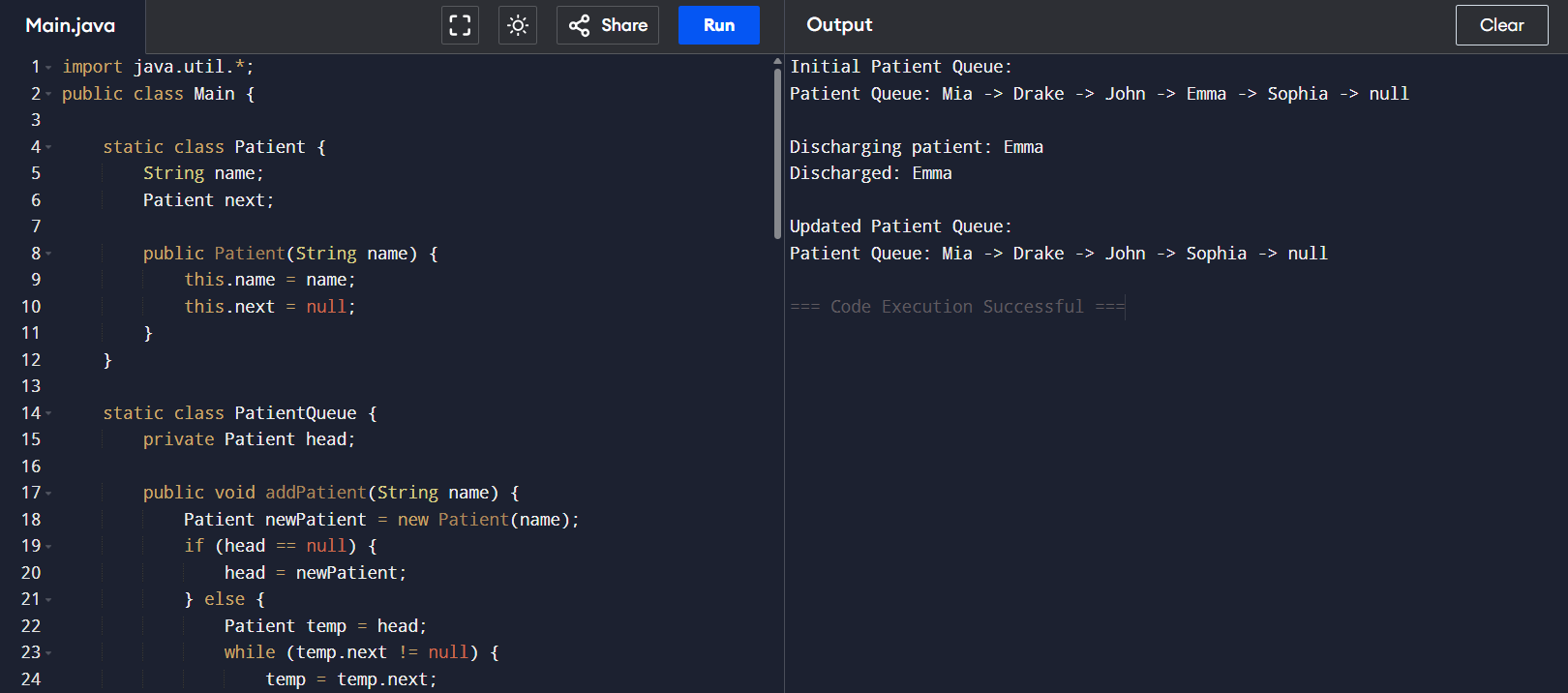
        queue.dischargePatient("Emma");

        System.out.println("\nUpdated Patient Queue:");

        queue.displayQueue();

    }

}



1. Browser History (Using Stack)

import java.util.Stack;

class BrowserHistory {

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

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

public void visit(String page) {

backStack.push(page);

forwardStack.clear();

System.out.println("Visited: " + page);

}

public void goBack() {

if (backStack.size() <= 1) {

System.out.println("No previous page.");

return;

}

String current = backStack.pop();

forwardStack.push(current);

System.out.println("Went back to: " + backStack.peek());

}

public void goForward() {

if (forwardStack.isEmpty()) {

System.out.println("No forward page.");

return;

}

String page = forwardStack.pop();

backStack.push(page);

System.out.println("Went forward to: " + page);

}

public void currentPage() {

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

}

public static void main(String[] args) {

BrowserHistory browser = new BrowserHistory();

browser.visit("google.com");

browser.visit("openai.com");

browser.visit("github.com");

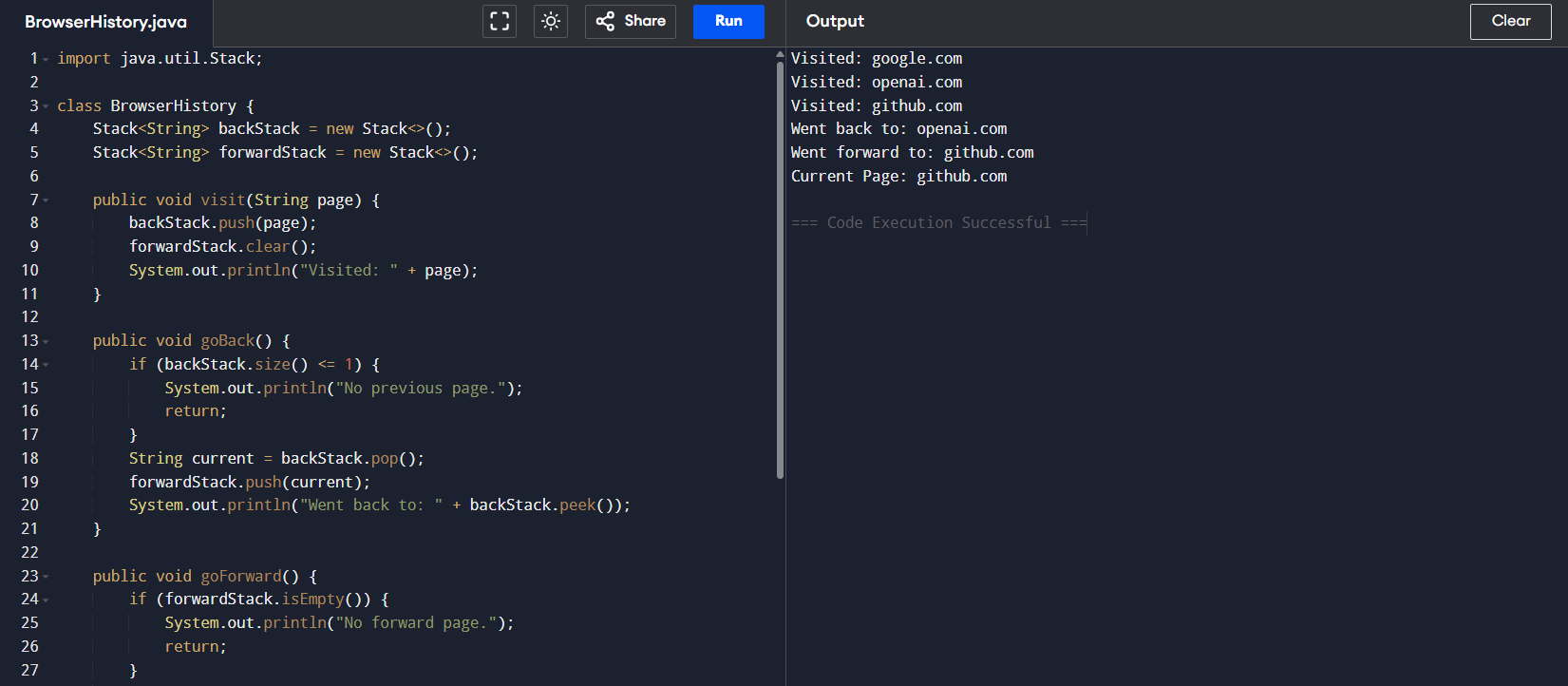
browser.goBack();

browser.goForward();

browser.currentPage();

}

}



1. Print Queue (Using LinkedList as Queue)

import java.util.LinkedList;

import java.util.Queue;

class PrintQueue {

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

public void addJob(String job) {

queue.offer(job);

System.out.println("Added: " + job);

}

public void processJob() {

if (queue.isEmpty()) {

System.out.println("No jobs to process.");

return;

}

System.out.println("Processing: " + queue.poll());

}

public void showPendingJobs() {

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

}

public static void main(String[] args) {

PrintQueue pq = new PrintQueue();

pq.addJob("File1.pdf");

pq.addJob("File2.docx");

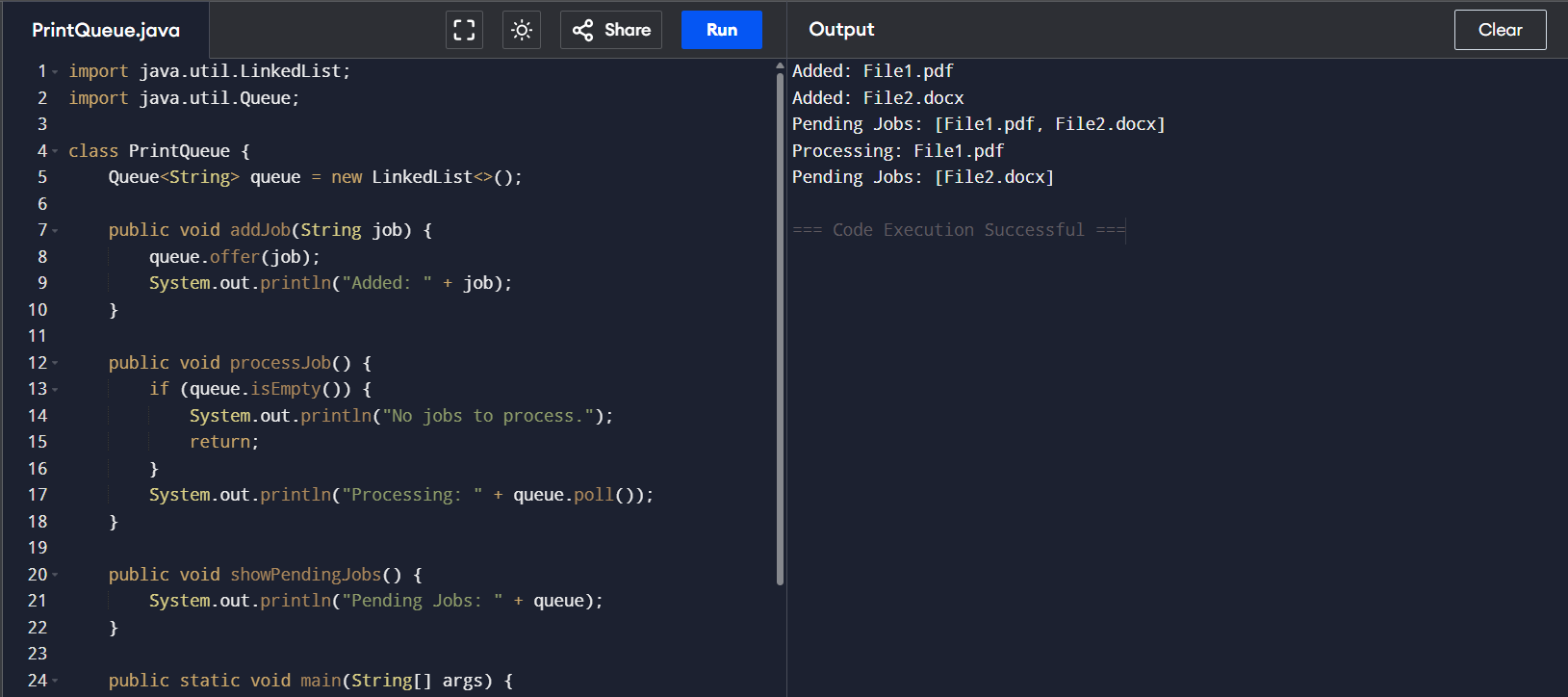
pq.showPendingJobs();

pq.processJob();

pq.showPendingJobs();

}

}



1. Hospital Bed Management (Using LinkedList)

import java.util.LinkedList;

class BedManagement {

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

public void assignBed(String patient) {

beds.add(patient);

System.out.println("Assigned bed to: " + patient);

}

public void discharge(String patient) {

if (beds.remove(patient)) {

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

} else {

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

}

}

public void showOccupancy() {

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

}

public static void main(String[] args) {

BedManagement bm = new BedManagement();

bm.assignBed("Alice");

bm.assignBed("Bob");

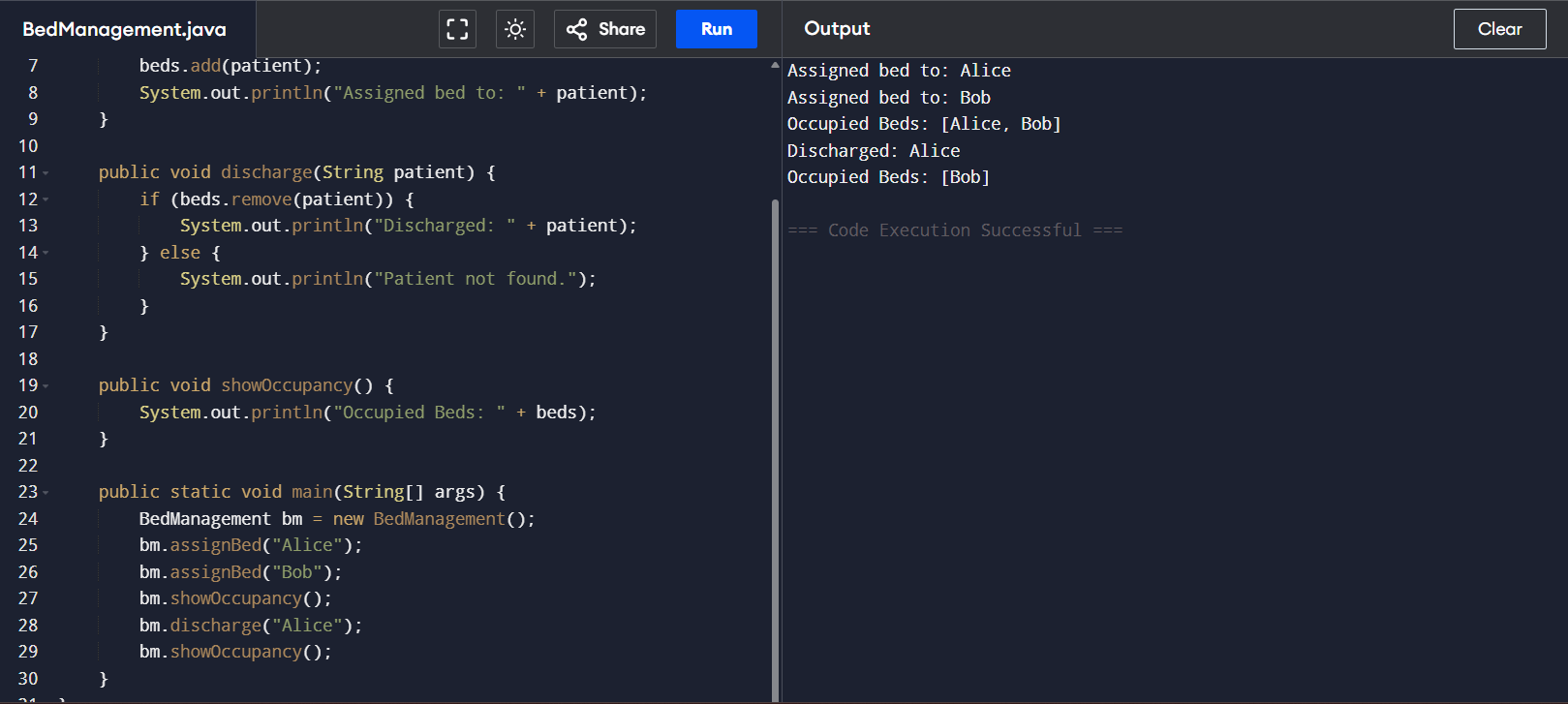
bm.showOccupancy();

bm.discharge("Alice");

bm.showOccupancy();

}

}



1. Undo-Redo Function (Using Stack)

import java.util.Stack;

class UndoRedo {

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

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

public void perform(String action) {

undoStack.push(action);

redoStack.clear();

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

}

public void undo() {

if (!undoStack.isEmpty()) {

String action = undoStack.pop();

redoStack.push(action);

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

} else {

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

}

}

public void redo() {

if (!redoStack.isEmpty()) {

String action = redoStack.pop();

undoStack.push(action);

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

} else {

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

}

}

public static void main(String[] args) {

UndoRedo ur = new UndoRedo();

ur.perform("Type A");

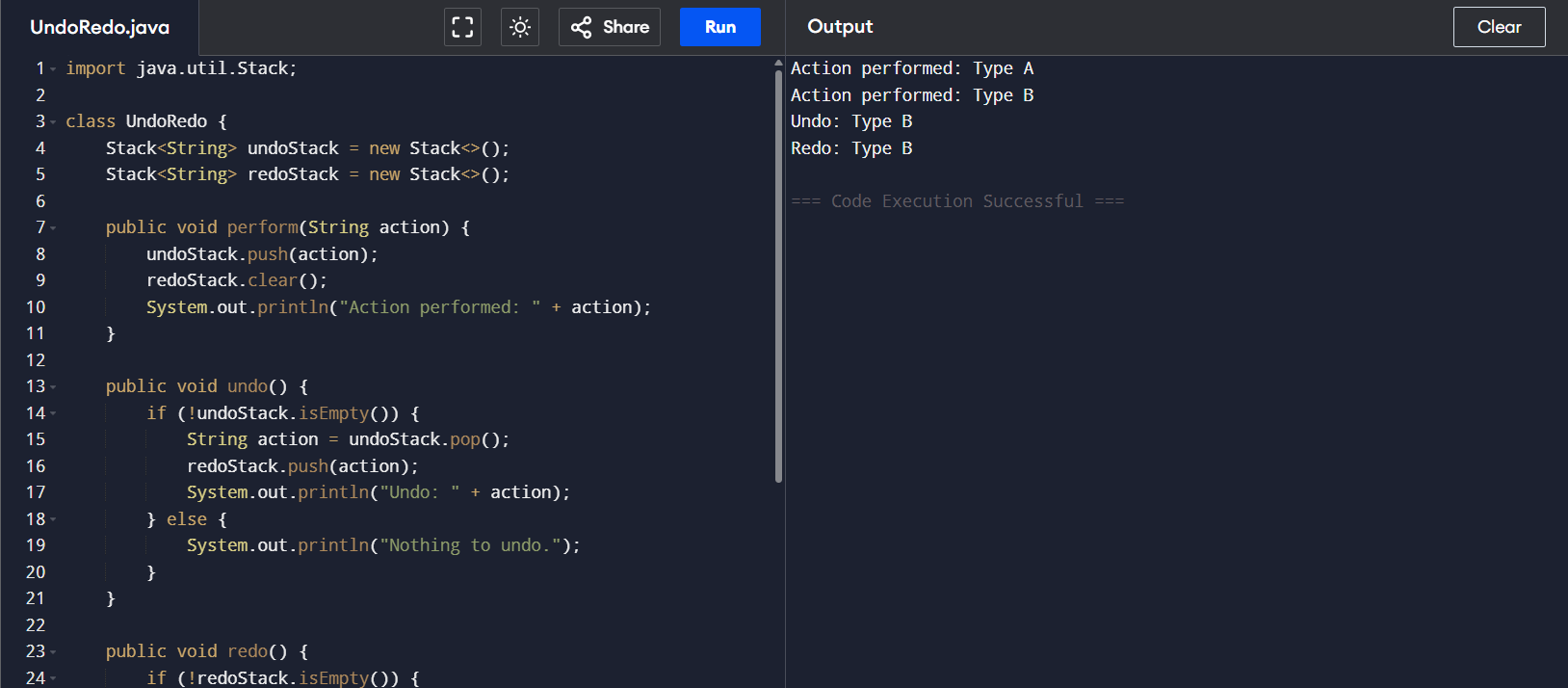
ur.perform("Type B");

ur.undo();

ur.redo();

}

}



1. Ticket Booking System (Using Queue)

import java.util.LinkedList;

import java.util.Queue;

class TicketBooking {

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

public void book(String person) {

queue.offer(person);

System.out.println("Added to queue: " + person);

}

public void serve() {

if (!queue.isEmpty()) {

System.out.println("Served: " + queue.poll());

} else {

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

}

}

public void cancel(String person) {

if (queue.remove(person)) {

System.out.println("Cancelled: " + person);

} else {

System.out.println("Not found in queue.");

}

}

public void showQueue() {

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

}

public static void main(String[] args) {

TicketBooking tb = new TicketBooking();

tb.book("Tom");

tb.book("Jerry");

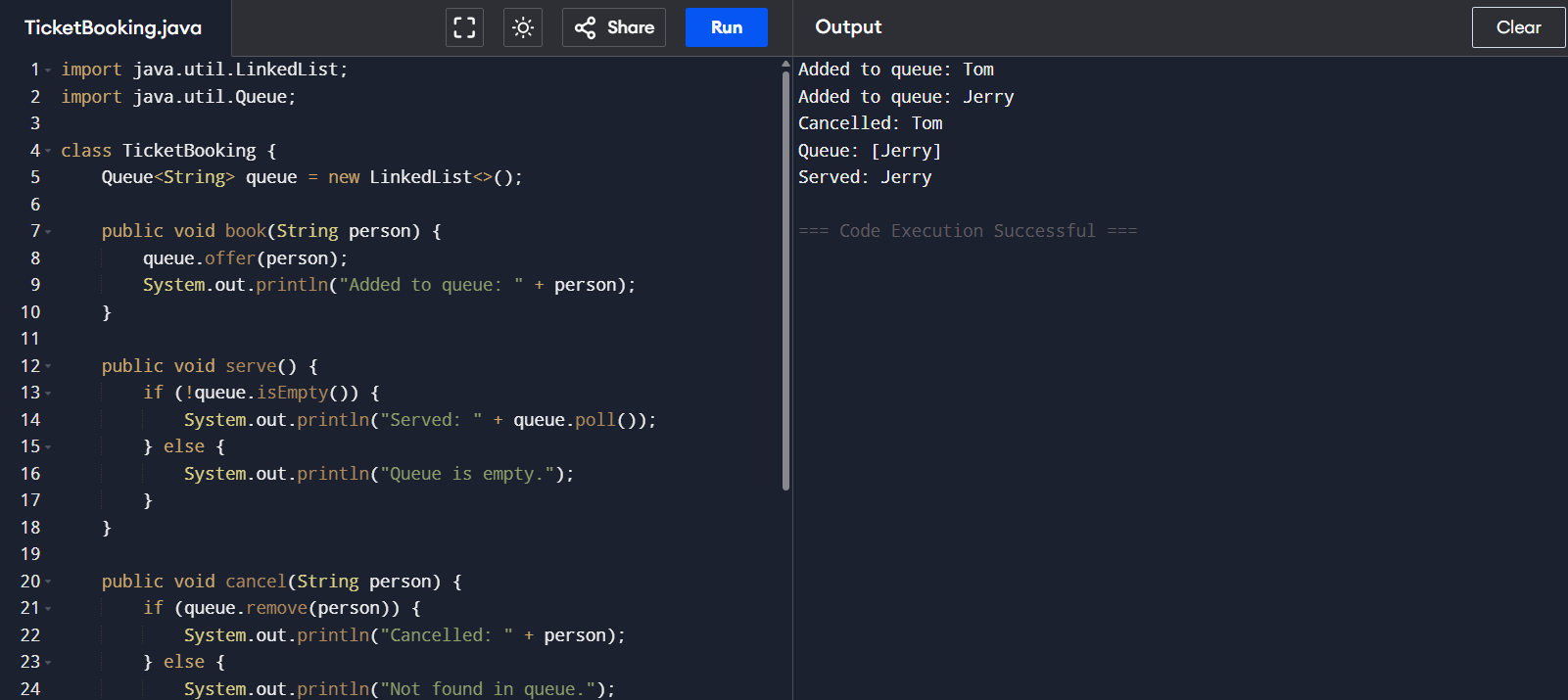
tb.cancel("Tom");

tb.showQueue();

tb.serve();

}

}



1. Car Wash Service Queue

import java.util.LinkedList;

class CarWash {

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

public void addNormalCar(String car) {

queue.addLast(car);

}

public void addVIPCar(String car) {

queue.addFirst(car);

}

public void washCar() {

if (!queue.isEmpty()) {

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

} else {

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

}

}

public void showQueue() {

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

}

public static void main(String[] args) {

CarWash cw = new CarWash();

cw.addNormalCar("Car1");

cw.addVIPCar("VIP1");

cw.addNormalCar("Car2");

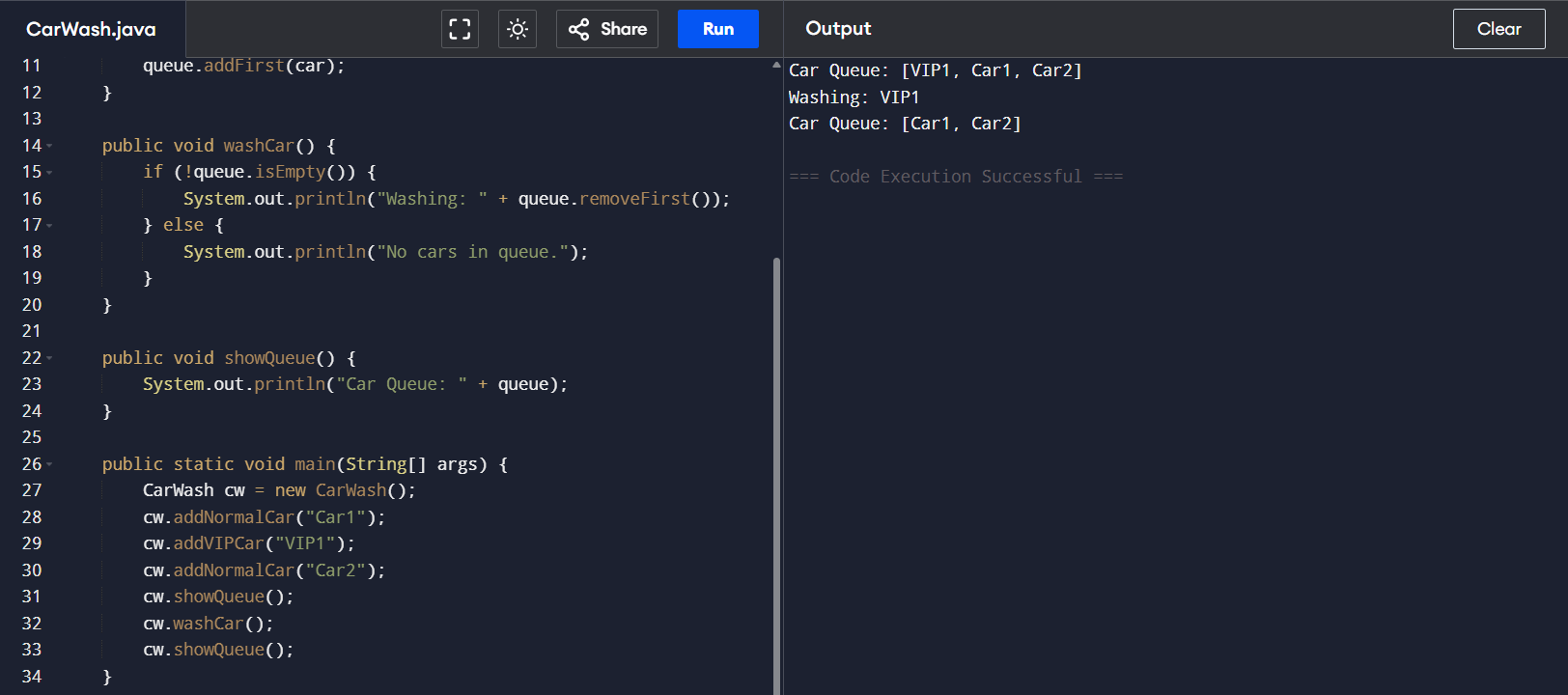
cw.showQueue();

cw.washCar();

cw.showQueue();

}

}



1. Library Book Stack (Using Stack)

import java.util.Stack;

class LibraryStack {

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

public void addBook(String book) {

books.push(book);

}

public void removeBook() {

if (!books.isEmpty()) {

System.out.println("Removed: " + books.pop());

} else {

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

}

}

public void peekTopBook() {

if (!books.isEmpty()) {

System.out.println("Top Book: " + books.peek());

} else {

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

}

}

public static void main(String[] args) {

LibraryStack ls = new LibraryStack();

ls.addBook("Java");

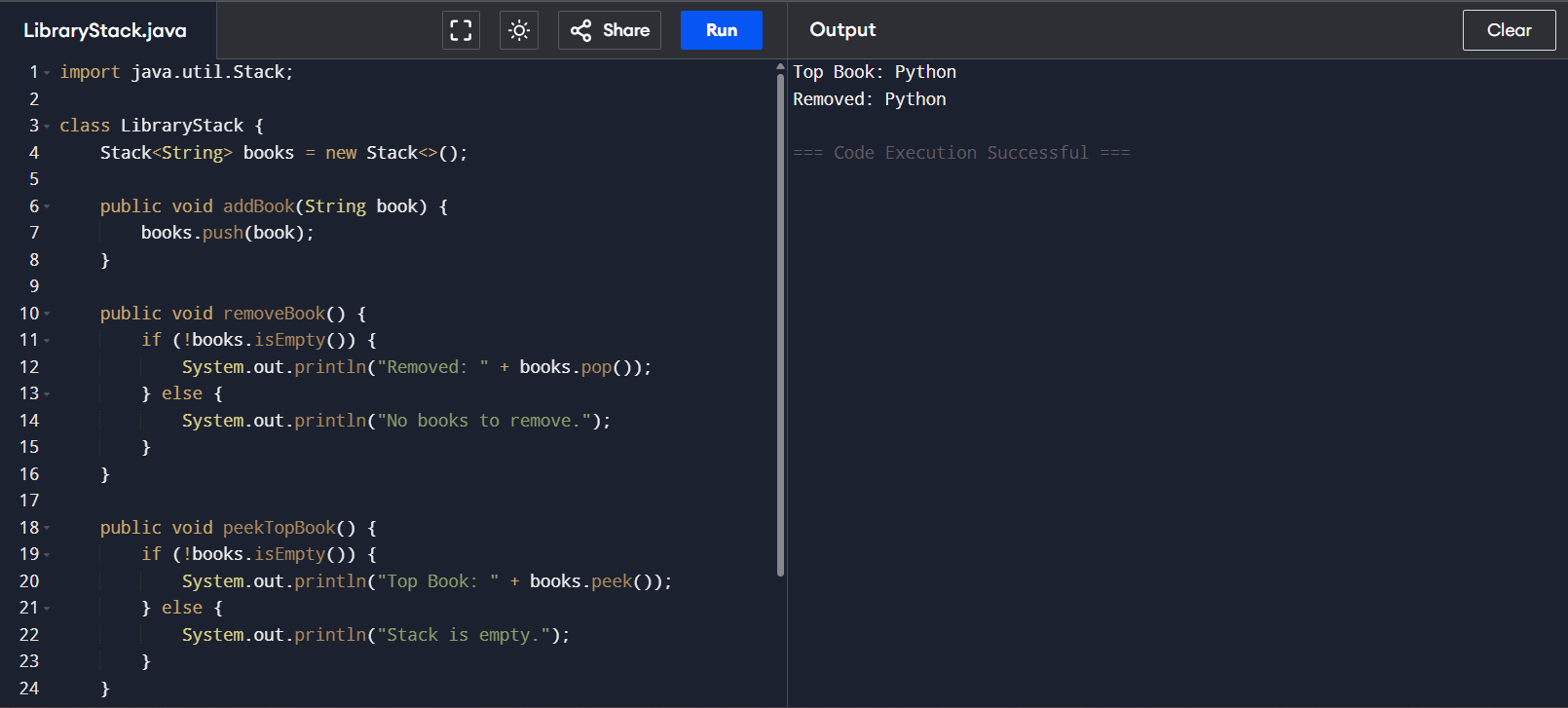
ls.addBook("Python");

ls.peekTopBook();

ls.removeBook();

}

}



1. Expression Evaluator (Infix to Postfix & Evaluate)

import java.util.Stack;

class ExpressionEvaluator {

public static int precedence(char op) {

switch (op) {

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

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

}

return -1;

}

public static String infixToPostfix(String exp) {

StringBuilder result = new StringBuilder();

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

for (char c : exp.toCharArray()) {

if (Character.isLetterOrDigit(c)) {

result.append(c);

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

stack.push(c);

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

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

result.append(stack.pop());

stack.pop();

} else {

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

result.append(stack.pop());

stack.push(c);

}

}

while (!stack.isEmpty())

result.append(stack.pop());

return result.toString();

}

public static int evaluatePostfix(String exp) {

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

for (char c : exp.toCharArray()) {

if (Character.isDigit(c)) {

stack.push(c - '0');

} else {

int b = stack.pop();

int a = stack.pop();

switch (c) {

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

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

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

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

}

}

}

return stack.pop();

}

public static void main(String[] args) {

String infix = "3+(2\*4)";

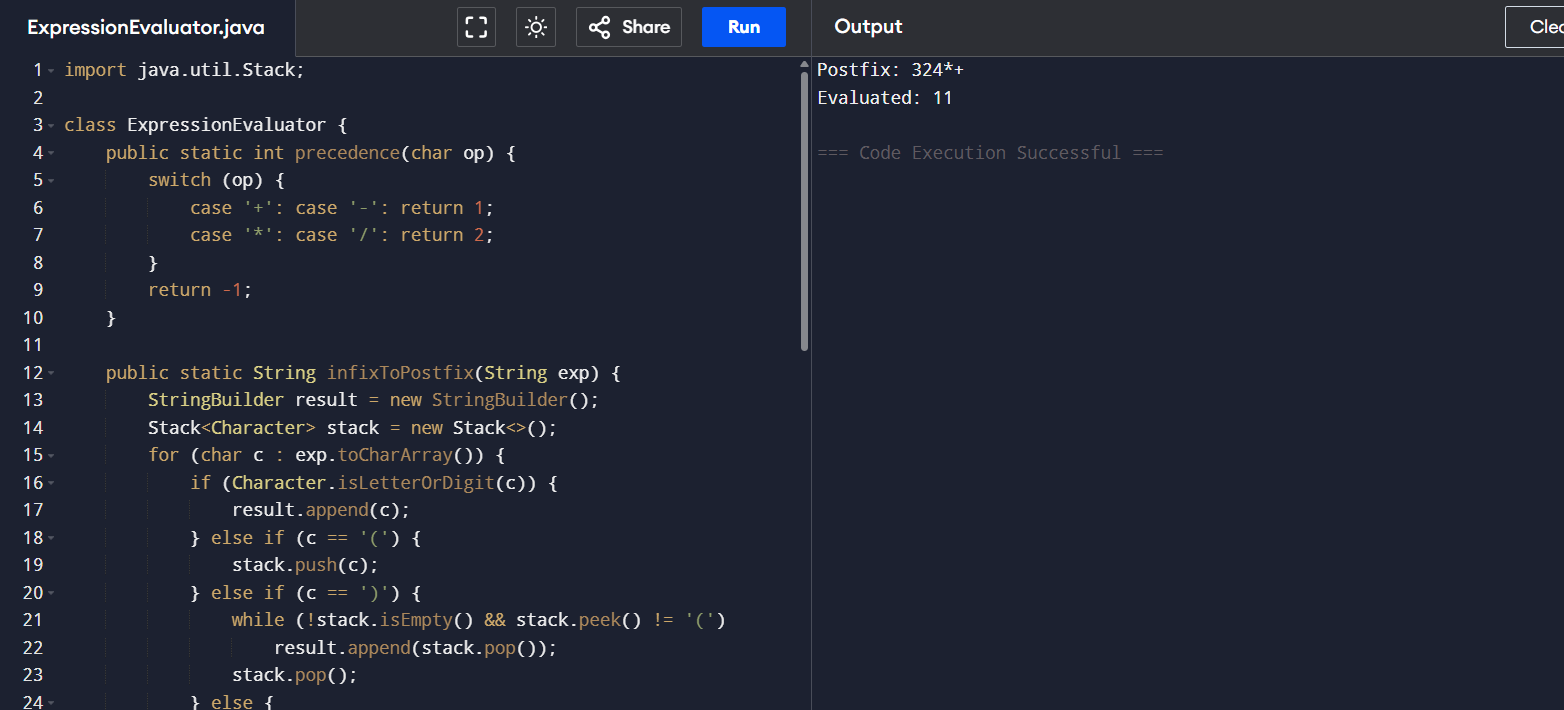
String postfix = infixToPostfix(infix);

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

System.out.println("Evaluated: " + evaluatePostfix(postfix));

}

}



1. Reverse Queue Using Stack

import java.util.\*;

class ReverseQueue {

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

public void enqueue(String name) {

queue.offer(name);

}

public void reverse() {

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

while (!queue.isEmpty())

stack.push(queue.poll());

while (!stack.isEmpty())

queue.offer(stack.pop());

}

public void showQueue() {

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

}

public static void main(String[] args) {

ReverseQueue rq = new ReverseQueue();

rq.enqueue("A");

rq.enqueue("B");

rq.enqueue("C");

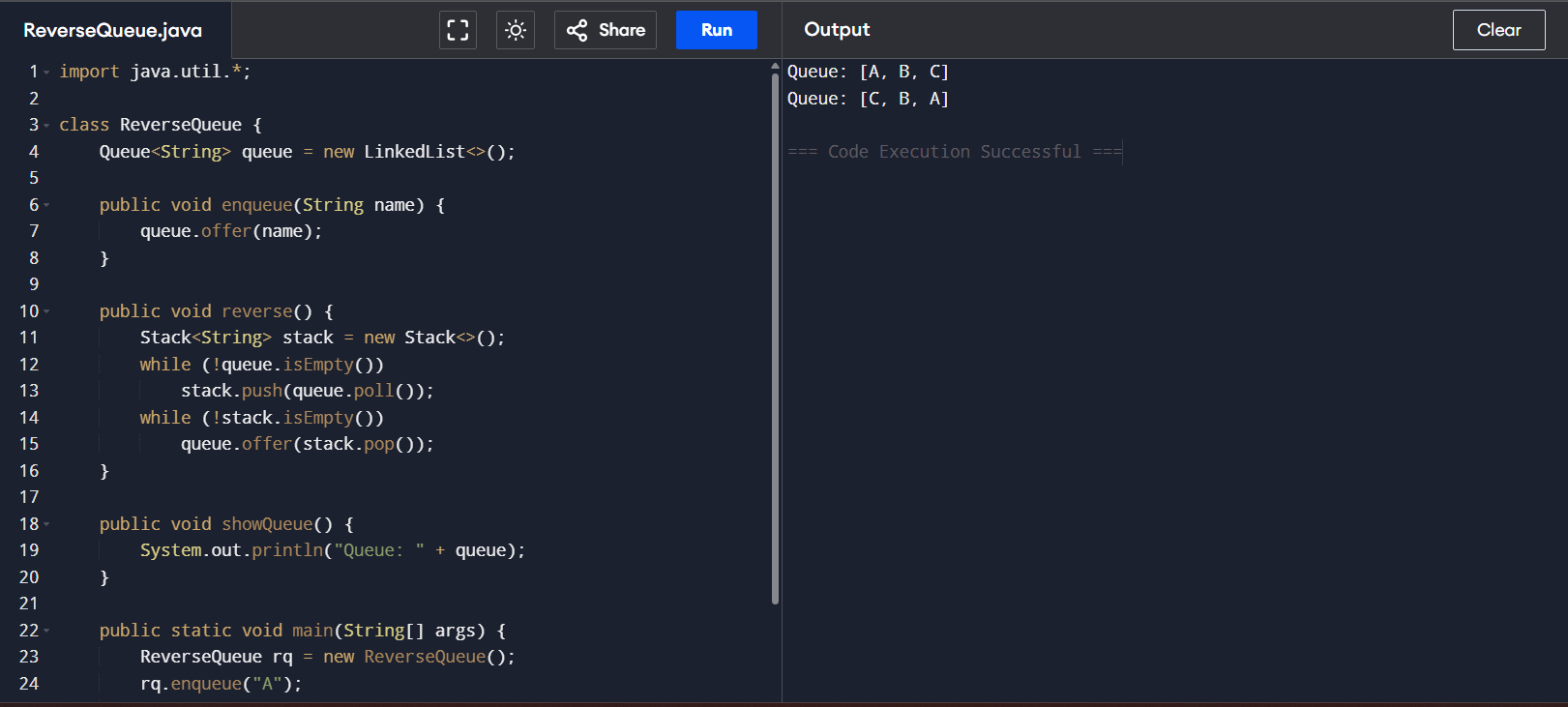
rq.showQueue();

rq.reverse();

rq.showQueue();

}

}



1. Student Admission Queue with Emergency Slot

import java.util.LinkedList;

class StudentAdmission {

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

public void addStudent(String student) {

queue.addLast(student);

}

public void addVIPStudent(String student) {

queue.addFirst(student);

}

public void admitStudent() {

if (!queue.isEmpty()) {

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

} else {

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

}

}

public void showQueue() {

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

}

public static void main(String[] args) {

StudentAdmission sa = new StudentAdmission();

sa.addStudent("Ravi");

sa.addVIPStudent("Priya");

sa.addStudent("Anil");

sa.showQueue();

sa.admitStudent();

sa.showQueue();

}

}

