Maintain a patient queue using a linked list where

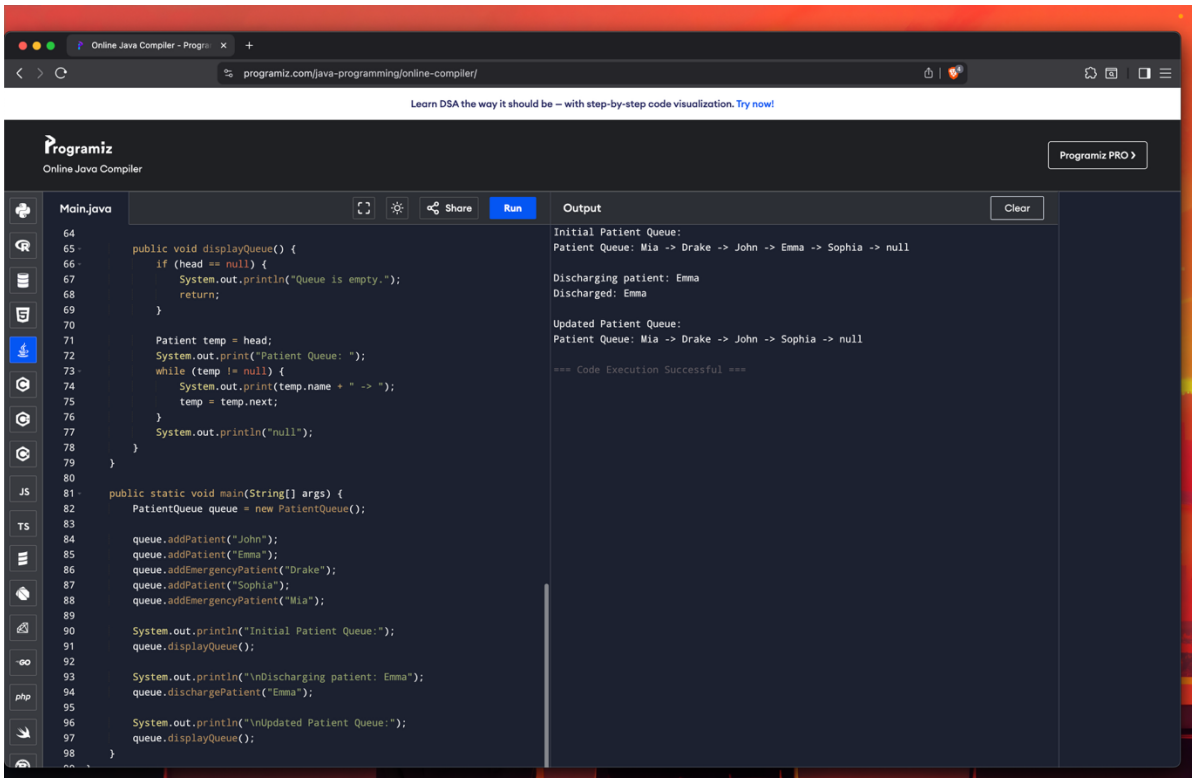
- Patients can be added (registered) at the end.
- Emergency patients should be added at the beginning.
- Support removing discharged patients.

```
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)

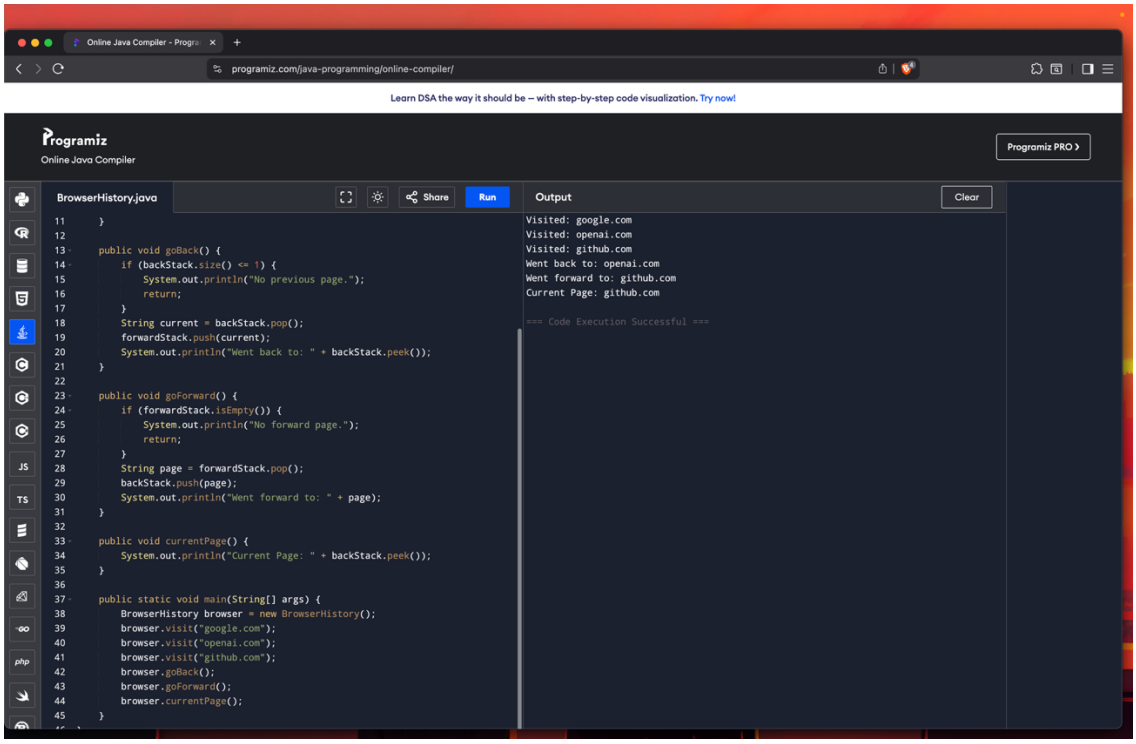
Scenario: Simulate a web browser's back and forward functionality using two stacks. Features:

Visit new page→ Push to `backStack`, clear `forwardStack`.

Go back → Pop from `backStack` and push to `forwardStack`.

Go forward → Pop from `forwardStack` and push to `backStack`.

Use two Stacks



2. Print Queue (Using LinkedList as Queue)

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

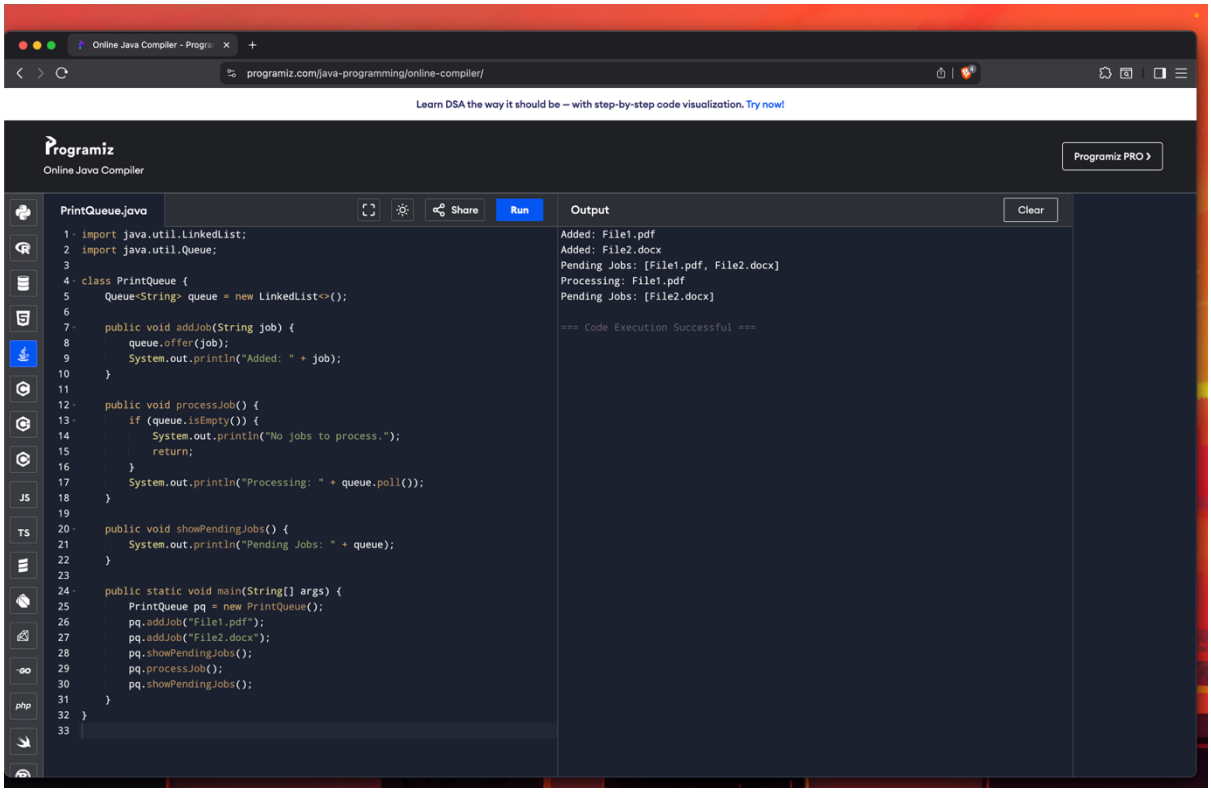
Features:

* Add new print jobs

* Process jobs in order

* View pending jobs

Use LinkedList as Queue



3. Hospital Bed Management (Using LinkedList)

Scenario: Track patients occupying hospital beds.

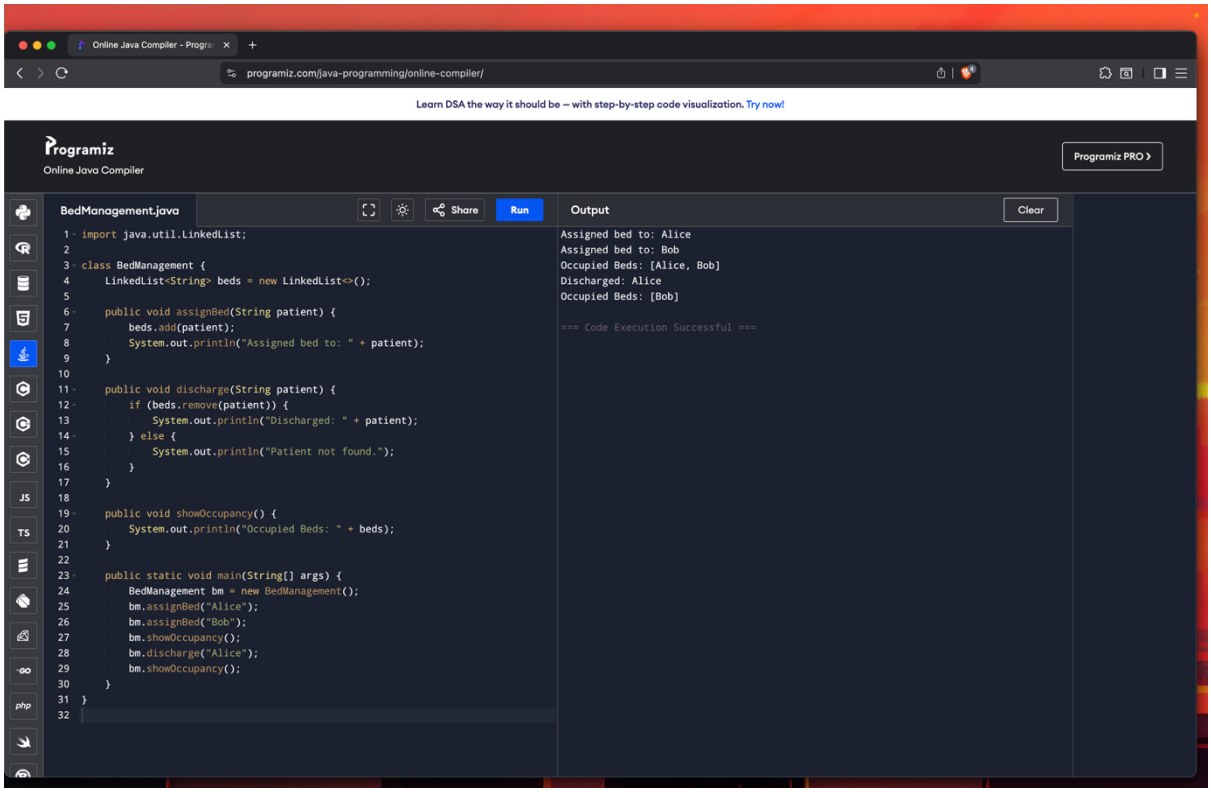
Features:

* Assign bed to new patient

* Discharge patient (remove by name or ID)

* Display current occupancy

Use LinkedList to represent beds



4. Undo-Redo Function (Using Stack)

Scenario: Track document edits with undo and redo.

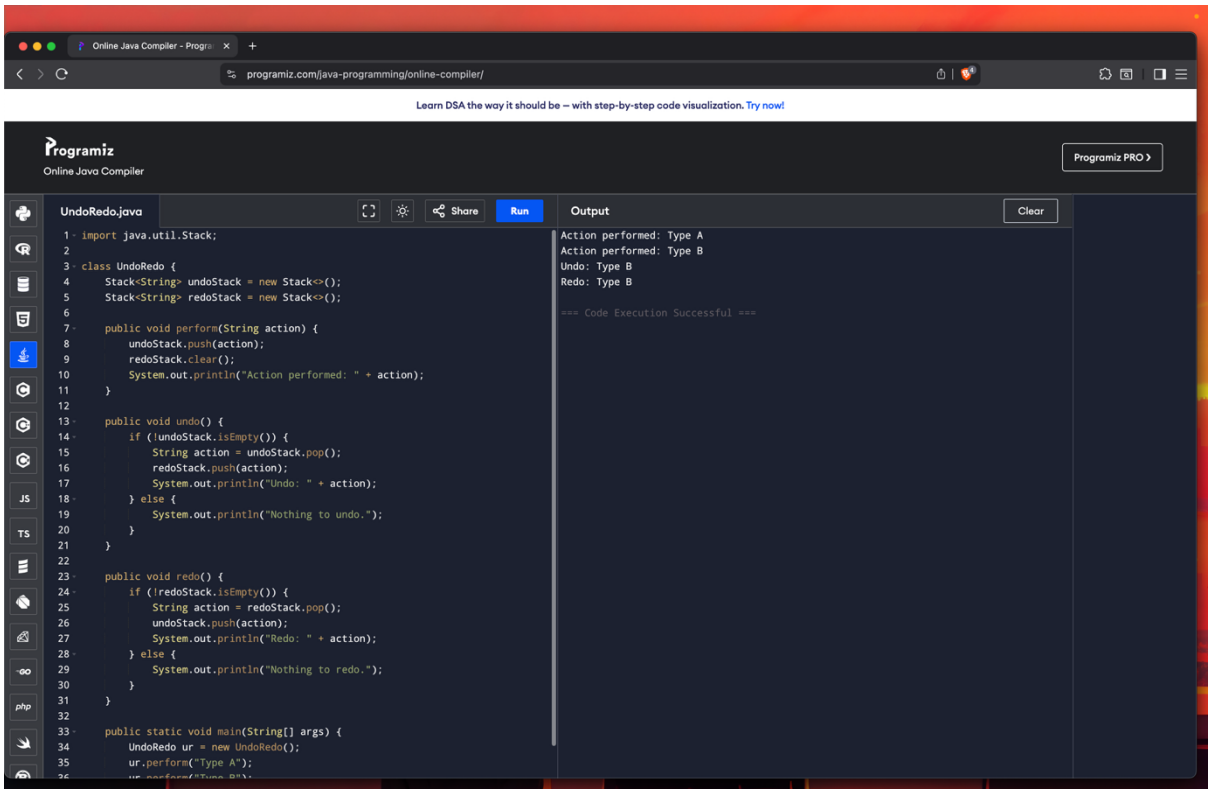
Features:

* Perform an action → Push to `undoStack`

* Undo → Move action to `redoStack`

* Redo → Move back to `undoStack`

Use two Stacks



5. Ticket Booking System (Using Queue)**

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

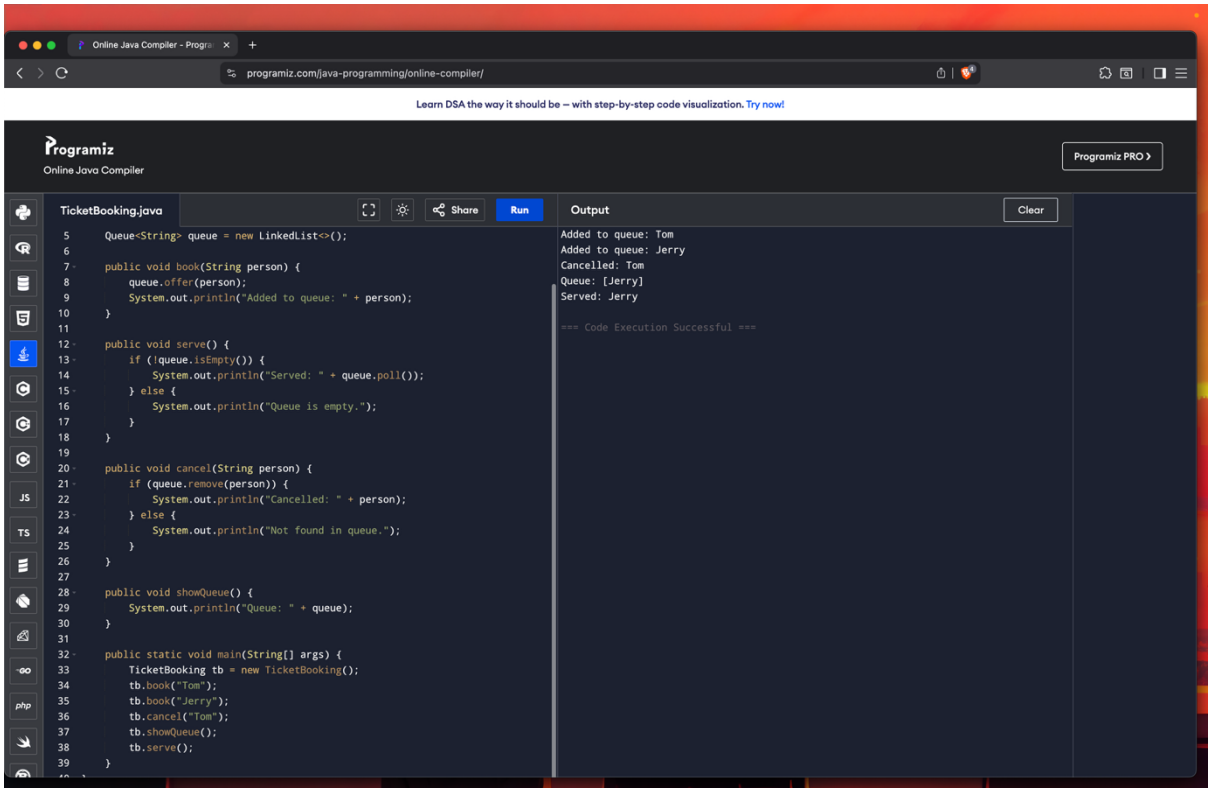
Features:

* Add person to booking queue

* Serve next person (dequeue)

* Cancel ticket (remove specific person)

Use Queue with LinkedList



6. Car Wash Service Queue

Scenario: Cars line up at a car wash center.

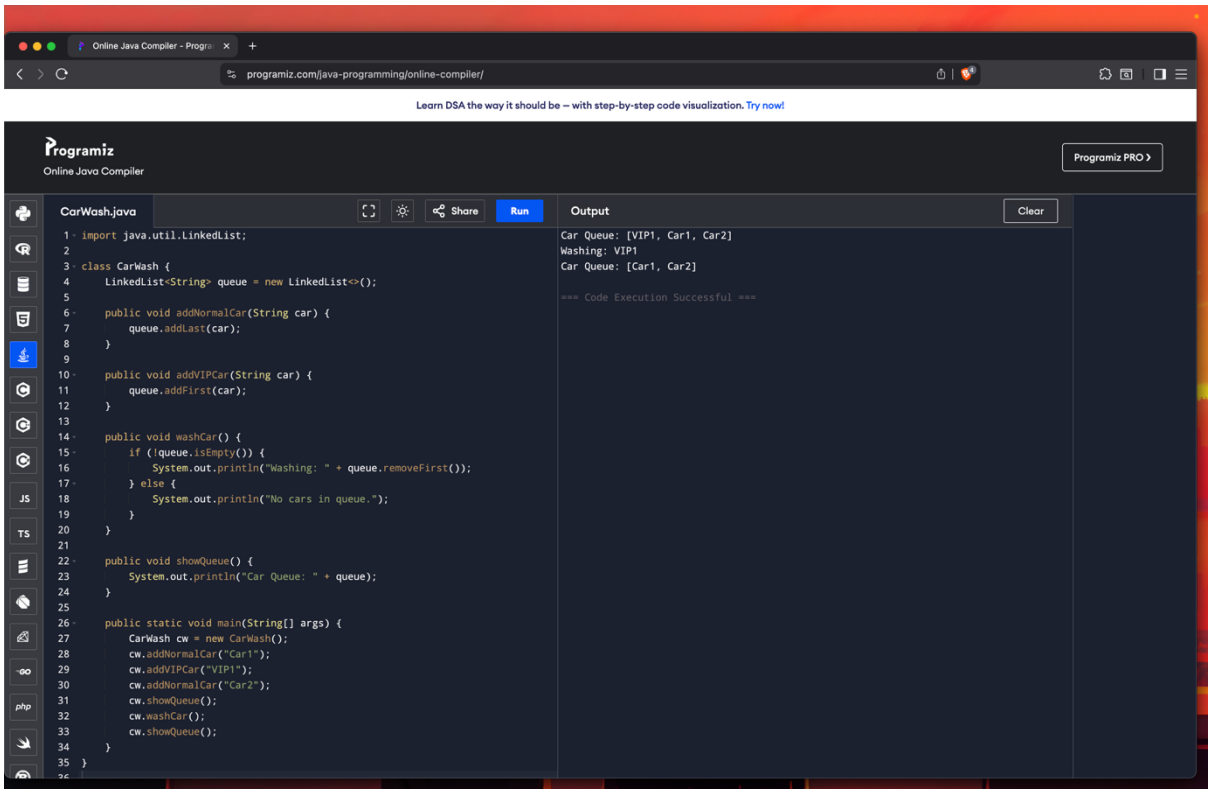
Features:

* Add normal cars to the end

* VIP cars go to the front

* Remove car after washing

Use LinkedList with front/back insertions



7. Library Book Stack (Using Stack)**

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

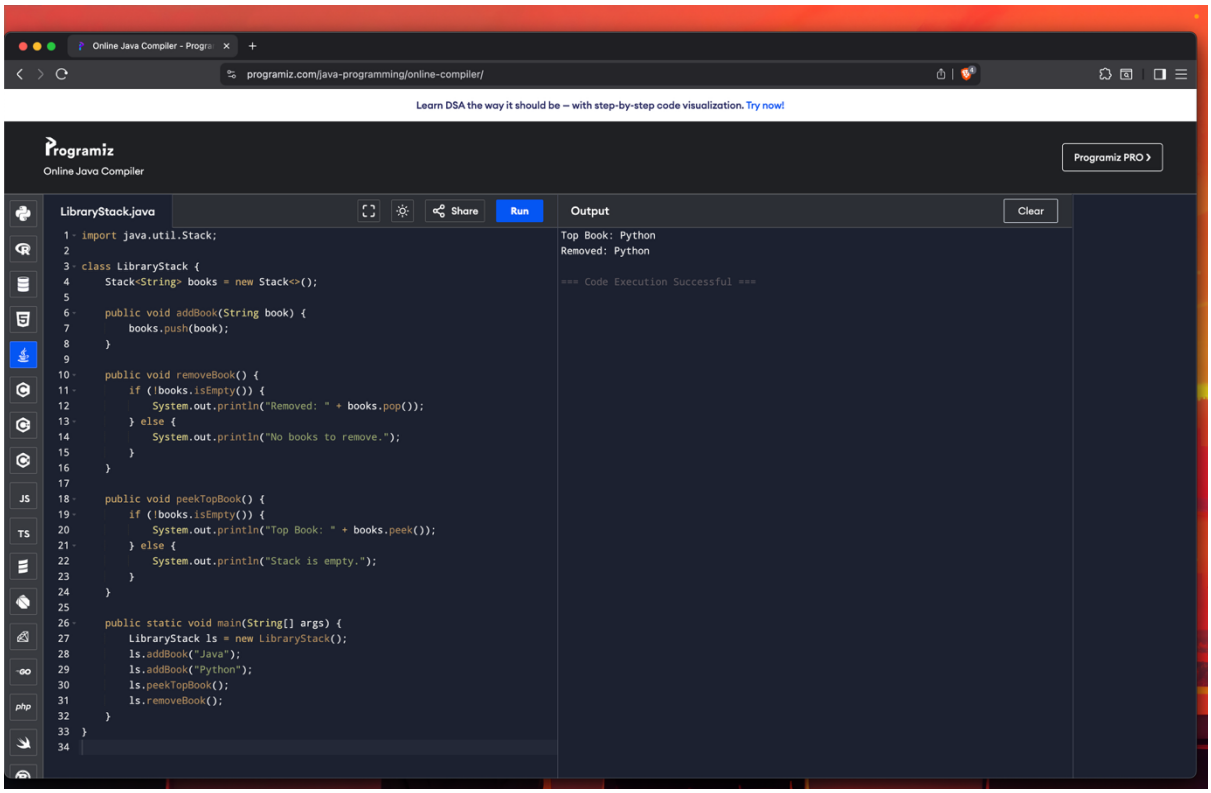
Features:

* Add book (push)

* Remove book (pop)

* Peek top book

Use Stack



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

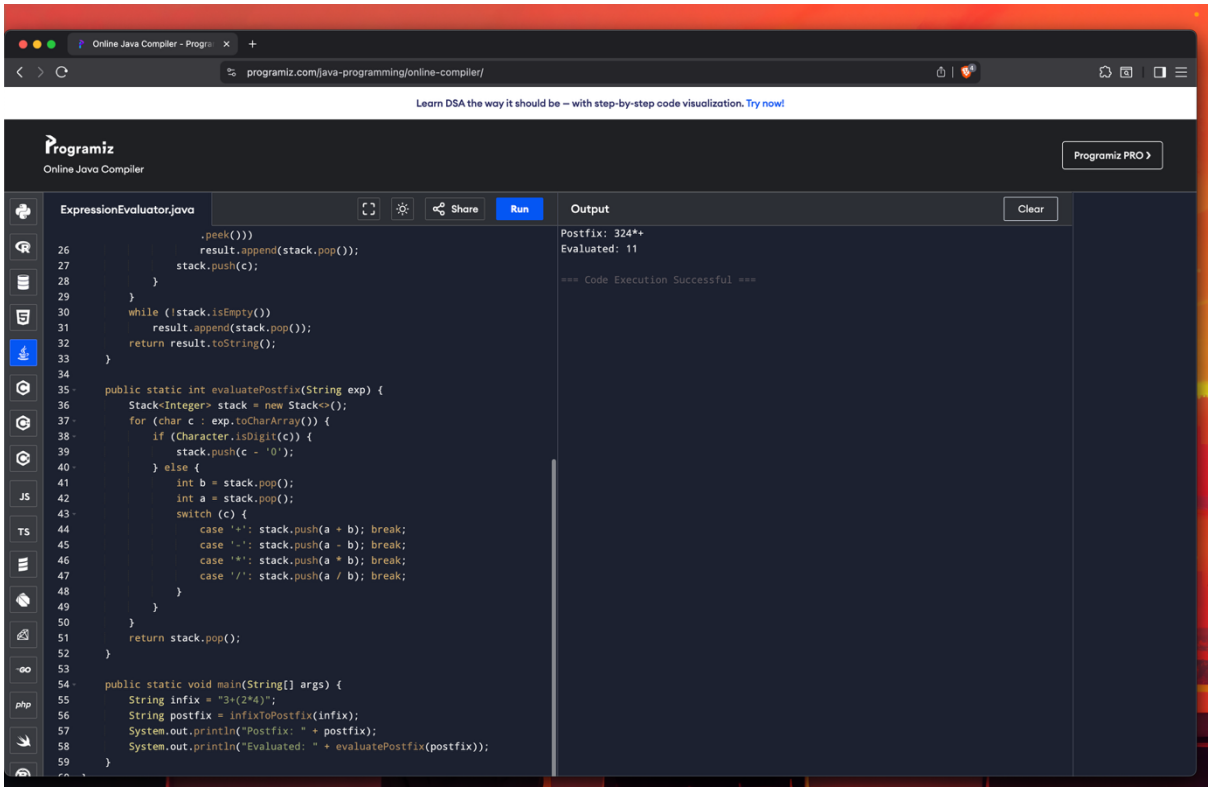
Scenario: Create a calculator to evaluate expressions.

Features:

* Convert infix to postfix

* Evaluate postfix using stack

Use Stack for operators and operands



9. Reverse Queue Using Stack**

Scenario: Reverse the order of a customer service queue.

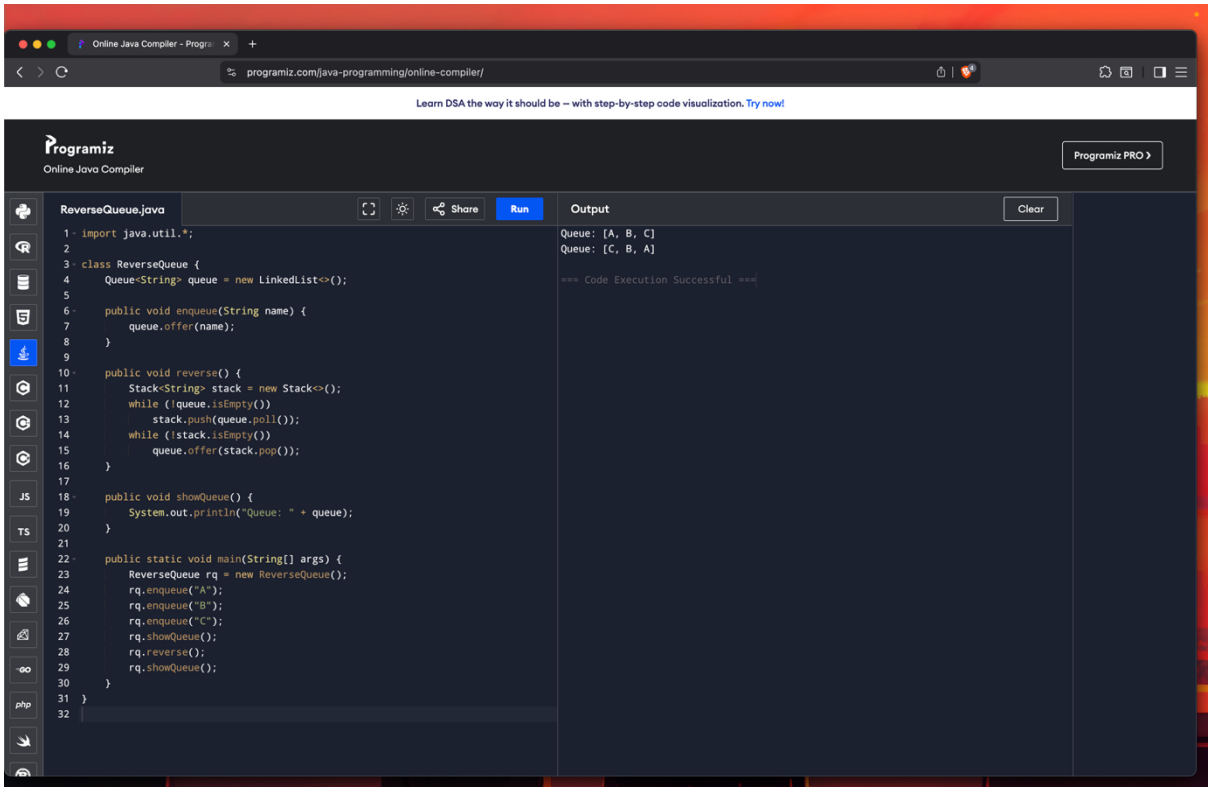
Features:

* Enqueue customers

* Reverse using stack

* Display new order

Use Queue + Stack



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

Use LinkedList