

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();
```

```
}
```

```
}
```

The screenshot shows the Programiz Online Java Compiler interface. The code editor on the left contains a Java program that implements a queue. The program starts by adding patients to the queue: John, Emma, Drake, Sophia, and Mia. It then prints the initial queue, discharges Emma, and prints the updated queue. The output on the right shows the execution results, confirming that Emma was discharged and the queue was updated correctly. The code is as follows:

```
64
65 public void displayQueue() {
66     if (head == null) {
67         System.out.println("Queue is empty.");
68         return;
69     }
70
71     Patient temp = head;
72     System.out.print("Patient Queue: ");
73     while (temp != null) {
74         System.out.print(temp.name + " -> ");
75         temp = temp.next;
76     }
77     System.out.println("null");
78 }
79
80
81 public static void main(String[] args) {
82     PatientQueue queue = new PatientQueue();
83
84     queue.addPatient("John");
85     queue.addPatient("Emma");
86     queue.addEmergencyPatient("Drake");
87     queue.addPatient("Sophia");
88     queue.addEmergencyPatient("Mia");
89
90     System.out.println("Initial Patient Queue:");
91     queue.displayQueue();
92
93     System.out.println("\nDischarging patient: Emma");
94     queue.dischargePatient("Emma");
95
96     System.out.println("\nUpdated Patient Queue:");
97     queue.displayQueue();
98 }
```

The output on the right shows the following results:

```
Initial Patient Queue:
Patient Queue: Mia -> Drake -> John -> Emma -> Sophia -> null

Discharging patient: Emma
Discharged: Emma

Updated Patient Queue:
Patient Queue: Mia -> Drake -> John -> Sophia -> null

=== Code Execution Successful ===
```

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

The screenshot shows the Programiz Online Java Compiler interface. The code editor contains the following Java code for `BrowserHistory.java`:

```
11 }
12
13 public void goBack() {
14     if (backStack.size() <= 1) {
15         System.out.println("No previous page.");
16         return;
17     }
18     String current = backStack.pop();
19     forwardStack.push(current);
20     System.out.println("Went back to: " + backStack.peek());
21 }
22
23 public void goForward() {
24     if (forwardStack.isEmpty()) {
25         System.out.println("No forward page.");
26         return;
27     }
28     String page = forwardStack.pop();
29     backStack.push(page);
30     System.out.println("Went forward to: " + page);
31 }
32
33 public void currentPage() {
34     System.out.println("Current Page: " + backStack.peek());
35 }
36
37 public static void main(String[] args) {
38     BrowserHistory browser = new BrowserHistory();
39     browser.visit("google.com");
40     browser.visit("openai.com");
41     browser.visit("github.com");
42     browser.goBack();
43     browser.goForward();
44     browser.currentPage();
45 }
```

The output window on the right shows the following execution results:

```
Visited: google.com
Visited: openai.com
Visited: github.com
Went back to: openai.com
Went forward to: github.com
Current Page: github.com

=== Code Execution Successful ===
```

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

The screenshot shows a web browser window with the URL `programiz.com/java-programming/online-compiler/`. The page header includes the Programiz logo and a "Programiz PRO" button. The main content area is divided into two panels: a code editor on the left and an output panel on the right.

The code editor displays the following Java code for `PrintQueue.java`:

```
1- import java.util.LinkedList;
2- import java.util.Queue;
3-
4- class PrintQueue {
5-     Queue<String> queue = new LinkedList<>();
6-
7-     public void addJob(String job) {
8-         queue.offer(job);
9-         System.out.println("Added: " + job);
10-    }
11-
12-    public void processJob() {
13-        if (queue.isEmpty()) {
14-            System.out.println("No jobs to process.");
15-            return;
16-        }
17-        System.out.println("Processing: " + queue.poll());
18-    }
19-
20-    public void showPendingJobs() {
21-        System.out.println("Pending Jobs: " + queue);
22-    }
23-
24-    public static void main(String[] args) {
25-        PrintQueue pq = new PrintQueue();
26-        pq.addJob("File1.pdf");
27-        pq.addJob("File2.docx");
28-        pq.showPendingJobs();
29-        pq.processJob();
30-        pq.showPendingJobs();
31-    }
32- }
33-
```

The output panel shows the following text:

```
Added: File1.pdf
Added: File2.docx
Pending Jobs: [File1.pdf, File2.docx]
Processing: File1.pdf
Pending Jobs: [File2.docx]

=== Code Execution Successful ===
```

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

The screenshot shows a web browser window with the URL `programiz.com/java-programming/online-compiler/`. The page title is "Online Java Compiler - Programiz". The main content area displays a Java program named `BedManagement.java` with the following code:

```
1- import java.util.LinkedList;
2-
3- class BedManagement {
4-     LinkedList<String> beds = new LinkedList<>();
5-
6-     public void assignBed(String patient) {
7-         beds.add(patient);
8-         System.out.println("Assigned bed to: " + patient);
9-     }
10-
11-     public void discharge(String patient) {
12-         if (beds.remove(patient)) {
13-             System.out.println("Discharged: " + patient);
14-         } else {
15-             System.out.println("Patient not found.");
16-         }
17-     }
18-
19-     public void showOccupancy() {
20-         System.out.println("Occupied Beds: " + beds);
21-     }
22-
23-     public static void main(String[] args) {
24-         BedManagement bm = new BedManagement();
25-         bm.assignBed("Alice");
26-         bm.assignBed("Bob");
27-         bm.showOccupancy();
28-         bm.discharge("Alice");
29-         bm.showOccupancy();
30-     }
31- }
32-
```

The output of the program is displayed on the right side of the compiler interface:

```
Assigned bed to: Alice
Assigned bed to: Bob
Occupied Beds: [Alice, Bob]
Discharged: Alice
Occupied Beds: [Bob]

=== Code Execution Successful ===
```

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

The screenshot shows the Programiz Online Java Compiler interface. The code editor displays the following Java code for an Undo-Redo function:

```
1- import java.util.Stack;
2
3- class UndoRedo {
4-     Stack<String> undoStack = new Stack<>();
5-     Stack<String> redoStack = new Stack<>();
6
7-     public void perform(String action) {
8-         undoStack.push(action);
9-         redoStack.clear();
10-        System.out.println("Action performed: " + action);
11-    }
12
13-    public void undo() {
14-        if (!undoStack.isEmpty()) {
15-            String action = undoStack.pop();
16-            redoStack.push(action);
17-            System.out.println("Undo: " + action);
18-        } else {
19-            System.out.println("Nothing to undo.");
20-        }
21-    }
22
23-    public void redo() {
24-        if (!redoStack.isEmpty()) {
25-            String action = redoStack.pop();
26-            undoStack.push(action);
27-            System.out.println("Redo: " + action);
28-        } else {
29-            System.out.println("Nothing to redo.");
30-        }
31-    }
32
33-    public static void main(String[] args) {
34-        UndoRedo ur = new UndoRedo();
35-        ur.perform("Type A");
36-        ur.undo();
37-        ur.redo();
38-    }
39-}
```

The output window on the right shows the following execution results:

```
Action performed: Type A
Action performed: Type B
Undo: Type B
Redo: Type B
=== Code Execution Successful ===
```

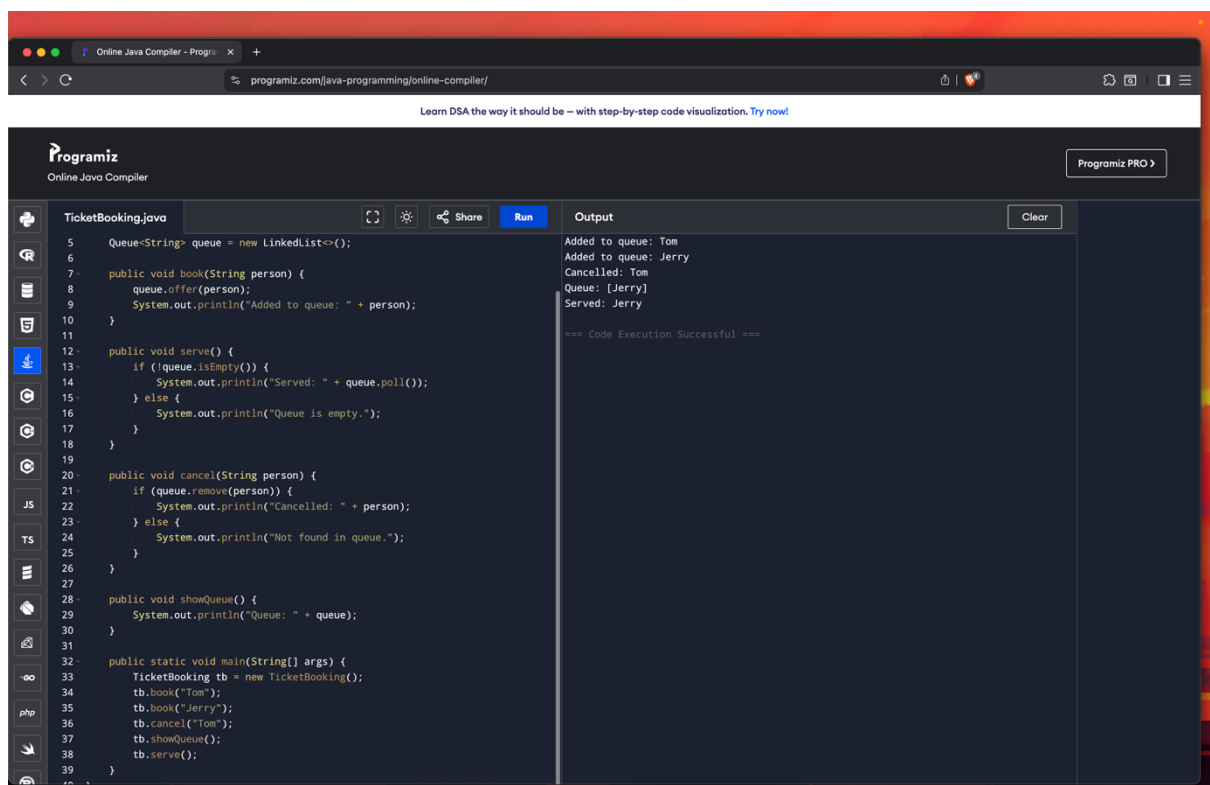

5. Ticket Booking System (Using Queue)**

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



The screenshot shows a web browser window with the URL `programiz.com/java-programming/online-compiler/`. The page title is "Online Java Compiler - Programiz". The main content area displays a Java code editor for a file named `TicketBooking.java`. The code implements a ticket booking system using a `Queue` (represented by `LinkedList`). The code includes methods for adding a person to the queue, serving the next person, and canceling a ticket. The output window on the right shows the execution results.

```
5 Queue<String> queue = new LinkedList<>();
6
7 public void book(String person) {
8     queue.offer(person);
9     System.out.println("Added to queue: " + person);
10 }
11
12 public void serve() {
13     if (!queue.isEmpty()) {
14         System.out.println("Served: " + queue.poll());
15     } else {
16         System.out.println("Queue is empty.");
17     }
18 }
19
20 public void cancel(String person) {
21     if (queue.remove(person)) {
22         System.out.println("Cancelled: " + person);
23     } else {
24         System.out.println("Not found in queue.");
25     }
26 }
27
28 public void showQueue() {
29     System.out.println("Queue: " + queue);
30 }
31
32 public static void main(String[] args) {
33     TicketBooking tb = new TicketBooking();
34     tb.book("Tom");
35     tb.book("Jerry");
36     tb.cancel("Tom");
37     tb.showQueue();
38     tb.serve();
39 }
```

Output:

```
Added to queue: Tom
Added to queue: Jerry
Cancelled: Tom
Queue: [Jerry]
Served: Jerry

=== Code Execution Successful ===
```

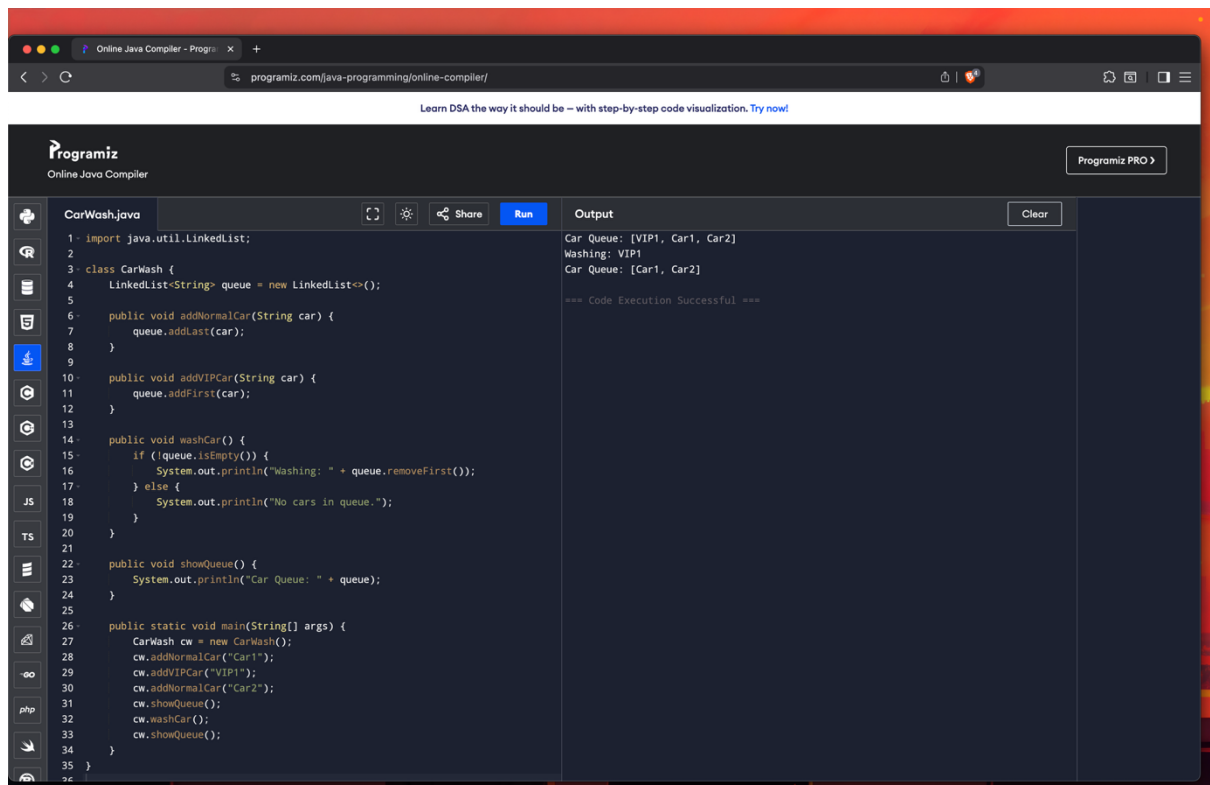
6. Car Wash Service Queue

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



The screenshot displays the Programiz Online Java Compiler interface. The code editor on the left contains the following Java code for CarWash.java:

```
1- import java.util.LinkedList;
2
3- class CarWash {
4-     LinkedList<String> queue = new LinkedList<>();
5
6-     public void addNormalCar(String car) {
7-         queue.addLast(car);
8-     }
9
10-    public void addVIPCar(String car) {
11-        queue.addFirst(car);
12-    }
13
14-    public void washCar() {
15-        if (!queue.isEmpty()) {
16-            System.out.println("Washing: " + queue.removeFirst());
17-        } else {
18-            System.out.println("No cars in queue.");
19-        }
20-    }
21
22-    public void showQueue() {
23-        System.out.println("Car Queue: " + queue);
24-    }
25
26-    public static void main(String[] args) {
27-        CarWash cw = new CarWash();
28-        cw.addNormalCar("Car1");
29-        cw.addVIPCar("VIP1");
30-        cw.addNormalCar("Car2");
31-        cw.showQueue();
32-        cw.washCar();
33-        cw.showQueue();
34-    }
35 }
```

The output panel on the right shows the following results:

```
Car Queue: [VIP1, Car1, Car2]
Washing: VIP1
Car Queue: [Car1, Car2]
=== Code Execution Successful ===
```

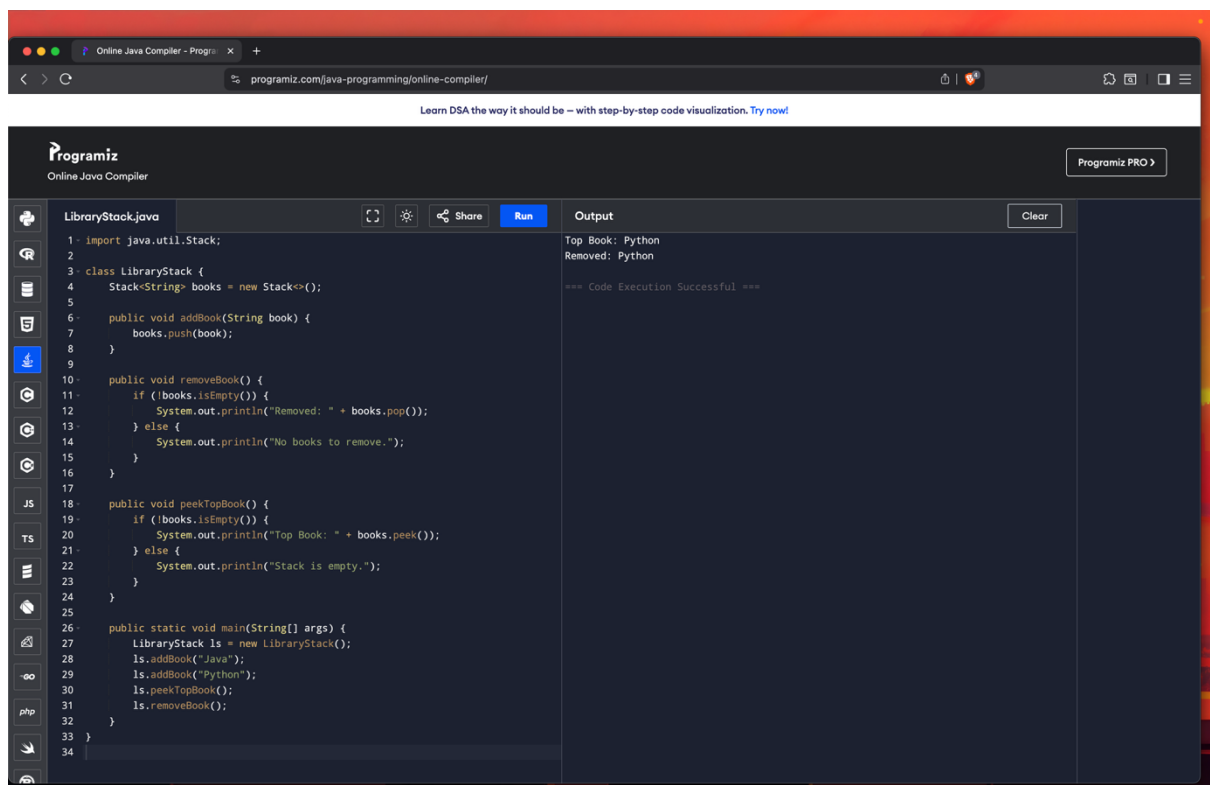
7. Library Book Stack (Using Stack)**

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

Features:

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

Use Stack



The screenshot shows the Programiz Online Java Compiler interface. The code editor contains a Java program named `LibraryStack.java` that implements a stack-based library system. The program includes methods for adding books, removing books, and peeking at the top book. The output window shows the results of the program execution.

```
1- import java.util.Stack;
2
3- class LibraryStack {
4-     Stack<String> books = new Stack<>();
5
6-     public void addBook(String book) {
7-         books.push(book);
8-     }
9
10-    public void removeBook() {
11-        if (!books.isEmpty()) {
12-            System.out.println("Removed: " + books.pop());
13-        } else {
14-            System.out.println("No books to remove.");
15-        }
16-    }
17
18-    public void peekTopBook() {
19-        if (!books.isEmpty()) {
20-            System.out.println("Top Book: " + books.peek());
21-        } else {
22-            System.out.println("Stack is empty.");
23-        }
24-    }
25
26-    public static void main(String[] args) {
27-        LibraryStack ls = new LibraryStack();
28-        ls.addBook("Java");
29-        ls.addBook("Python");
30-        ls.peekTopBook();
31-        ls.removeBook();
32-    }
33 }
34
```

Output:

```
Top Book: Python
Removed: Python
=== Code Execution Successful ===
```

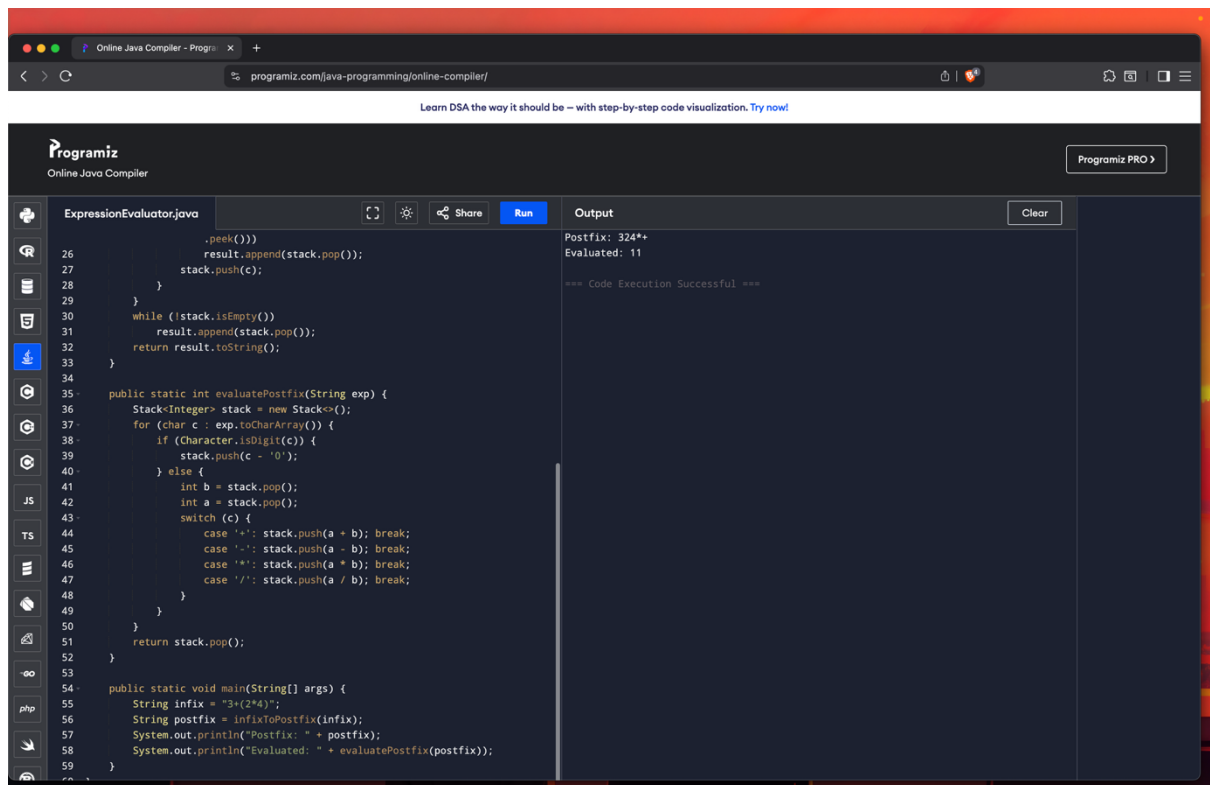
*8. Expression Evaluator (Infix to Postfix & Evaluate)

Scenario: Create a calculator to evaluate expressions.

Features:

- * Convert infix to postfix
- * Evaluate postfix using stack

Use Stack for operators and operands



The screenshot displays the Programiz Online Java Compiler interface. The main editor shows the code for `ExpressionEvaluator.java`. The code includes a `main` method that takes an infix expression `"3+(2*4)"`, converts it to postfix, and then evaluates it. The `evaluatePostfix` method uses a stack to process the postfix expression. The output panel shows the postfix expression `324*+` and the evaluated result `11`. The code execution was successful.

```
ExpressionEvaluator.java
26         .peek()))
27         result.append(stack.pop());
28         stack.push(c);
29     }
30     while (!stack.isEmpty())
31         result.append(stack.pop());
32     return result.toString();
33 }
34
35 public static int evaluatePostfix(String exp) {
36     Stack<Integer> stack = new Stack<>();
37     for (char c : exp.toCharArray()) {
38         if (Character.isDigit(c)) {
39             stack.push(c - '0');
40         } else {
41             int b = stack.pop();
42             int a = stack.pop();
43             switch (c) {
44                 case '+': stack.push(a + b); break;
45                 case '-': stack.push(a - b); break;
46                 case '*': stack.push(a * b); break;
47                 case '/': stack.push(a / b); break;
48             }
49         }
50     }
51     return stack.pop();
52 }
53
54 public static void main(String[] args) {
55     String infix = "3+(2*4)";
56     String postfix = infixToPostfix(infix);
57     System.out.println("Postfix: " + postfix);
58     System.out.println("Evaluated: " + evaluatePostfix(postfix));
59 }
```

Output

Postfix: 324*+
Evaluated: 11
=== Code Execution Successful ===

9. Reverse Queue Using Stack**

Scenario: Reverse the order of a customer service queue.

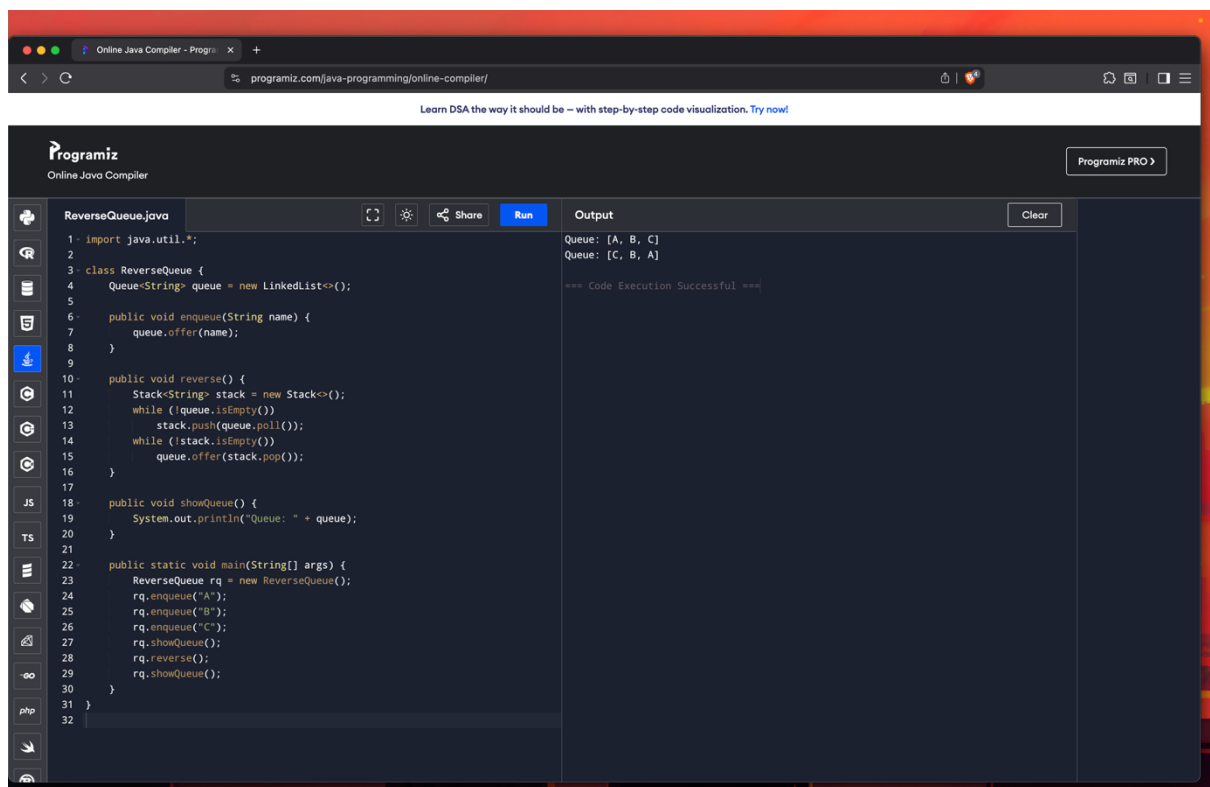
Features:

- * Enqueue customers

- * Reverse using stack

- * Display new order

Use Queue + Stack



The screenshot shows a web browser window with the URL `programiz.com/java-programming/online-compiler/`. The page title is "Online Java Compiler - Programiz". The main content area displays a Java code editor with the following code:

```
1- import java.util.*;
2-
3- class ReverseQueue {
4-     Queue<String> queue = new LinkedList<>();
5-
6-     public void enqueue(String name) {
7-         queue.offer(name);
8-     }
9-
10-    public void reverse() {
11-        Stack<String> stack = new Stack<>();
12-        while (!queue.isEmpty()) {
13-            stack.push(queue.poll());
14-            while (!stack.isEmpty()) {
15-                queue.offer(stack.pop());
16-            }
17-        }
18-
19-        public void showQueue() {
20-            System.out.println("Queue: " + queue);
21-        }
22-
23-        public static void main(String[] args) {
24-            ReverseQueue rq = new ReverseQueue();
25-            rq.enqueue("A");
26-            rq.enqueue("B");
27-            rq.enqueue("C");
28-            rq.showQueue();
29-            rq.reverse();
30-            rq.showQueue();
31-        }
32-    }
33- }
```

The output section on the right shows the following text:

```
Queue: [A, B, C]
Queue: [C, B, A]
=== Code Execution Successful ===
```

10. Student Admission Queue with Emergency Slot

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

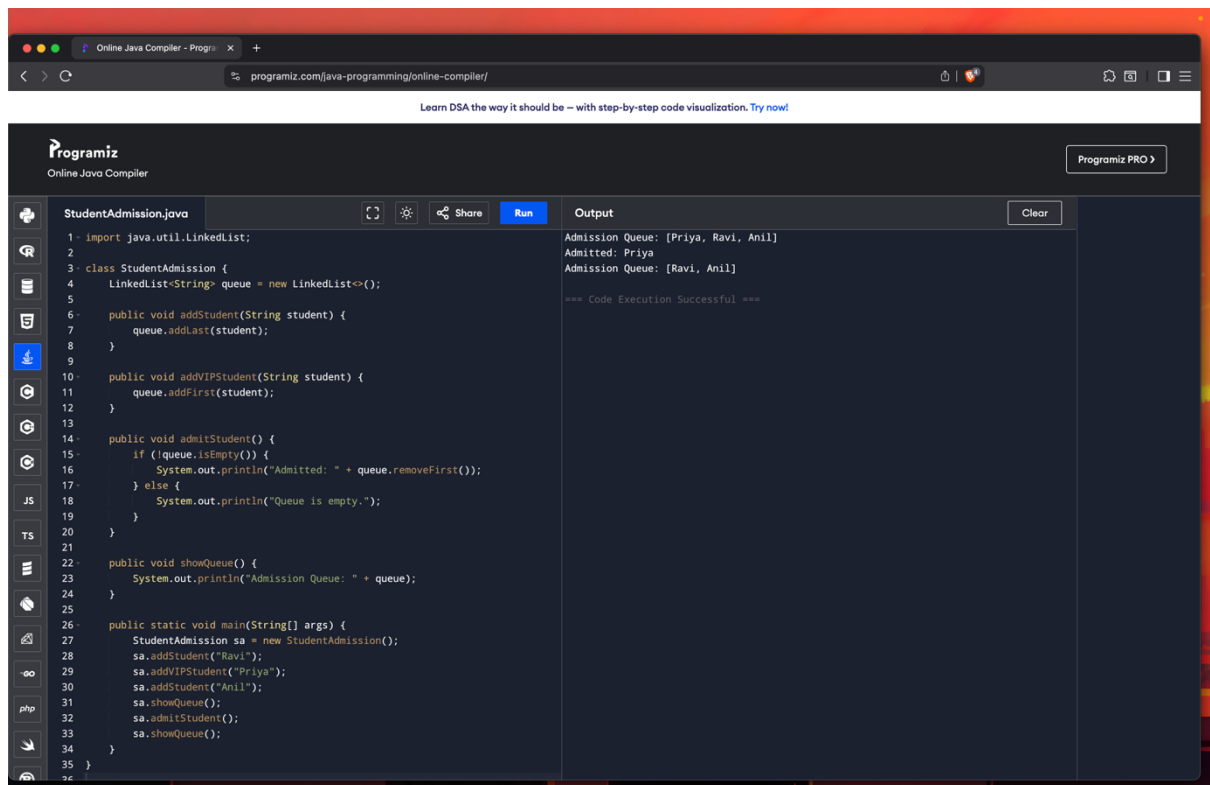
Features:

- * Add student normally (end)

- * Add VIP (front)

- * Remove admitted student

Use LinkedList



```
StudentAdmission.java
1- import java.util.LinkedList;
2-
3- class StudentAdmission {
4-     LinkedList<String> queue = new LinkedList<>();
5-
6-     public void addStudent(String student) {
7-         queue.addLast(student);
8-     }
9-
10-    public void addVIPStudent(String student) {
11-        queue.addFirst(student);
12-    }
13-
14-    public void admitStudent() {
15-        if (!queue.isEmpty()) {
16-            System.out.println("Admitted: " + queue.removeFirst());
17-        } else {
18-            System.out.println("Queue is empty.");
19-        }
20-    }
21-
22-    public void showQueue() {
23-        System.out.println("Admission Queue: " + queue);
24-    }
25-
26-    public static void main(String[] args) {
27-        StudentAdmission sa = new StudentAdmission();
28-        sa.addStudent("Ravi");
29-        sa.addVIPStudent("Priya");
30-        sa.addStudent("Anil");
31-        sa.showQueue();
32-        sa.admitStudent();
33-        sa.showQueue();
34-    }
35-}
```

Output

```
Admission Queue: [Priya, Ravi, Anil]
Admitted: Priya
Admission Queue: [Ravi, Anil]
=== Code Execution Successful ===
```

