**Data Structures — Stack and Queue-Practice Problems (Any Four) Practice Problem 1. Undo Feature in a Text Editor (Stack)**

**Concept:** Stack (LIFO) — Undo operations

**Scenario:**

You’re developing a simple text editor. Every time a user types a word, it’s stored. When the user presses **Undo**, the last word typed should be removed.

**Boilerplate Code:**

**import *java*.*util*.*\**;**

***public class* TextEditorUndo {**

***public static* void *main*(String[] args) { Stack<String> stack = *new* Stack<>(); Scanner sc = *new Scanner*(System.*in*); *while* (true) {**

**System.*out*.*print*("Enter command (TYPE <word>/UNDO/PRINT/EXIT): ");**

**String cmd = sc.*next*();**

***// TODO: Handle TYPE - push word***

***// TODO: Handle UNDO - pop last word // TODO: Handle PRINT - display full sentence // TODO: Handle EXIT - break***

**}**

**}**

**}**

**Example Input/Output:**

Commands:

TYPE Hello

TYPE World

UNDO

TYPE Java

PRINT

Output:

Hello Java

**Hint:**

Use a stack to store typed words. Pop words when “UNDO” is triggered.

PROGRAM

import java.util.\*;

public class TextEditorUndo {

public static void main(String[] args) {

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

Scanner sc = new Scanner(System.in);

while (true) {

System.out.print("Enter command (TYPE <word>/UNDO/PRINT/EXIT): ");

String cmd = sc.next();

if (cmd.equalsIgnoreCase("TYPE")) {

String word = sc.next();

stack.push(word);

}

else if (cmd.equalsIgnoreCase("UNDO")) {

if (!stack.isEmpty()) {

stack.pop();

} else {

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

}

}

else if (cmd.equalsIgnoreCase("PRINT")) {

if (stack.isEmpty()) {

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

} else {

for (String s : stack) {

System.out.print(s + " ");

}

System.out.println();

}

}

else if (cmd.equalsIgnoreCase("EXIT")) {

break;

}

else {

System.out.println("Invalid command!");

}

}

sc.close();

}

}

OUTPUT

TYPE Namaste

TYPE Bharat

UNDO

TYPE Java

PRINT

Output: Namaste Java

**Practice Problem 2. Browser Navigation Simulation (Stack)**

**Concept:** Stack (LIFO) — Back/Forward navigation

**Scenario:**

Simulate how a web browser works — when you open new pages, they’re added to a stack. When you click **Back**, you go to the previous page.

**Boilerplate Code:**

**import *java*.*util*.*\**;**

***public class* BrowserNavigation {**

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

**Stack<String> backStack = *new* Stack<>(); Stack<String> forwardStack = *new* Stack<>(); Scanner sc = *new Scanner*(System.*in*);**

**String current = "Home";**

***while* (true) {**

**System.*out*.*print*("Command**

**(VISIT/BACK/FORWARD/PRINT/EXIT): ");**

**String cmd = sc.*next*();**

***// TODO: Implement VISIT - push to backStack***

***// TODO: Implement BACK - move from backStack to forwardStack***

***// TODO: Implement FORWARD - move from forwardStack to backStack***

***// TODO: PRINT current page* }**

**}**

**}**

**Example:**

Input:

VISIT google.com

VISIT youtube.com

VISIT github.com

BACK

BACK

PRINT

Output:

Current Page: google.com

**Hint:**

Use one stack for “visited pages”, another for “forward pages”.

PROGRAM

import java.util.\*;

public class BrowserNavigation {

public static void main(String[] args) {

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

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

Scanner sc = new Scanner(System.in);

String current = "Home";

while (true) {

System.out.print("Command (VISIT/BACK/FORWARD/PRINT/EXIT): ");

String cmd = sc.next();

if (cmd.equalsIgnoreCase("VISIT")) {

String page = sc.next();

backStack.push(current);

current = page;

forwardStack.clear();

}

else if (cmd.equalsIgnoreCase("BACK")) {

if (!backStack.isEmpty()) {

forwardStack.push(current);

current = backStack.pop();

} else {

System.out.println("No previous pages!");

}

}

else if (cmd.equalsIgnoreCase("FORWARD")) {

if (!forwardStack.isEmpty()) {

backStack.push(current);

current = forwardStack.pop();

} else {

System.out.println("No forward pages!");

}

}

else if (cmd.equalsIgnoreCase("PRINT")) {

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

}

else if (cmd.equalsIgnoreCase("EXIT")) {

break;

}

else {

System.out.println("Invalid command!");

}

}

sc.close();

}

}

OUTPUT

VISIT google.in

VISIT srmist.edu.in

VISIT iitmadras.ac.in

BACK

BACK

PRINT

Output: Current Page: google.in

**Practice Problem 3. Print Queue System for Office Printer (Queue)**

**Concept:** Queue (FIFO) — Job Scheduling

**Scenario:**

In an office, multiple users send print jobs.

The printer executes them **in the order they were received**.

**Boilerplate Code:**

import *java*.*util*.*\**;

*public class* PrintQueueSystem {

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

Queue<String> printQueue = *new* LinkedList<>(); Scanner sc = *new Scanner*(System.*in*);

*while* (true) {

System.*out*.*print*("Command (ADD <doc>/PRINT/EXIT): ");

String cmd = sc.*next*();

*// TODO: Handle ADD - enqueue document // TODO: Handle PRINT - dequeue and show printed document*

*// TODO: Handle EXIT - break loop*

}

}

}

**Example Input:**

ADD Document1

ADD Document2

PRINT

PRINT

PRINT

**Output:**

Printing Document1

Printing Document2

No jobs left!

**Hint:** Use a queue to enqueue and dequeue job names.

PROGRAM

import java.util.\*;

public class PrintQueueSystem {

public static void main(String[] args) {

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

Scanner sc = new Scanner(System.in);

while (true) {

System.out.print("Command (ADD <doc>/PRINT/EXIT): ");

String cmd = sc.next();

if (cmd.equalsIgnoreCase("ADD")) {

String doc = sc.next();

printQueue.add(doc);

}

else if (cmd.equalsIgnoreCase("PRINT")) {

if (!printQueue.isEmpty()) {

System.out.println("Printing " + printQueue.poll());

} else {

System.out.println("No jobs left!");

}

}

else if (cmd.equalsIgnoreCase("EXIT")) {

break;

}

else {

System.out.println("Invalid command!");

}

}

sc.close();

}

}

OUTPUT

ADD AadhaarForm

ADD PANForm

PRINT

PRINT

PRINT

Output:

Printing AadhaarForm

Printing PANForm

No jobs left!

**Practice Problem 4. Customer Service Counter Simulation (Queue)**

**Concept:** Queue (FIFO) — Real-life waiting line

**Scenario:**

At a bank counter, customers are served one after another in the order they arrive. When a customer is served, they leave the queue.

**Boilerplate Code:**

import *java*.*util*.*\**;

*public class* CustomerServiceCounter {

*public static* void *main*(String[] args) { Queue<String> queue = *new* LinkedList<>(); Scanner sc = *new Scanner*(System.*in*);

*while* (true) {

System.*out*.*print*("Command (ARRIVE <name>/SERVE/STATUS/EXIT): ");

String cmd = sc.*next*();

*// TODO: Handle ARRIVE - add to queue // TODO: Handle SERVE - remove from queue // TODO: Handle STATUS - display waiting list // TODO: Handle EXIT - break*

}

}

}

**Example Input:**

ARRIVE Alice

ARRIVE Bob

ARRIVE Charlie

SERVE

SERVE

STATUS

**Output:**

Serving Alice

Serving Bob

Waiting: [Charlie]

**Hint:**

Model each customer as a string and maintain the waiting list using a queue.

PROGRAM

import java.util.\*;

public class CustomerServiceCounter {

public static void main(String[] args) {

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

Scanner sc = new Scanner(System.in);

while (true) {

System.out.print("Command (ARRIVE <name>/SERVE/STATUS/EXIT): ");

String cmd = sc.next();

if (cmd.equalsIgnoreCase("ARRIVE")) {

String name = sc.next();

queue.add(name);

}

else if (cmd.equalsIgnoreCase("SERVE")) {

if (!queue.isEmpty()) {

System.out.println("Serving " + queue.poll());

} else {

System.out.println("No customers waiting!");

}

}

else if (cmd.equalsIgnoreCase("STATUS")) {

if (queue.isEmpty()) {

System.out.println("No one is waiting!");

} else {

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

}

}

else if (cmd.equalsIgnoreCase("EXIT")) {

break;

}

else {

System.out.println("Invalid command!");

}

}

sc.close();

}

}

OUTPUT

ARRIVE Aarav

ARRIVE Priya

ARRIVE Rohan

SERVE

SERVE

STATUS

Output:

Serving Aarav

Serving Priya

Waiting: [Rohan]