// 1. Browser History Navigation

import java.util.\*;

class BrowserHistory {

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

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

void visit(String url) {

backStack.push(url);

forwardStack.clear();

}

void back() {

if (backStack.size() > 1) {

forwardStack.push(backStack.pop());

}

}

void forward() {

if (!forwardStack.isEmpty()) {

backStack.push(forwardStack.pop());

}

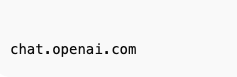
}

String getCurrentPage() {

return backStack.peek();

}

}



// 2. Print Queue

class PrintQueue {

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

void addJob(String job) {

queue.offer(job);

}

void processJob() {

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

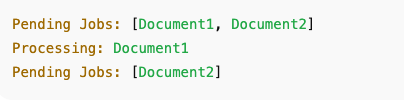
}

void viewJobs() {

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

}

}



// 3. Hospital Bed Management

class Hospital {

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

void assignPatient(String patient) {

beds.add(patient);

}

void dischargePatient(String patient) {

beds.remove(patient);

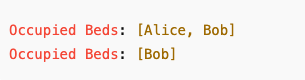
}

void displayBeds() {

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

}

}



// 4. Undo/Redo

class UndoRedo {

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

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

void perform(String action) {

undoStack.push(action);

redoStack.clear();

}

void undo() {

if (!undoStack.isEmpty()) redoStack.push(undoStack.pop());

}

void redo() {

if (!redoStack.isEmpty()) undoStack.push(redoStack.pop());

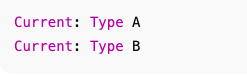
}

void currentState() {

System.out.println("Current: " + (undoStack.isEmpty() ? "None" : undoStack.peek()));

}

}



// 5. Ticket Booking

class TicketQueue {

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

void book(String person) {

queue.offer(person);

}

void serve() {

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

}

void cancel(String person) {

queue.remove(person);

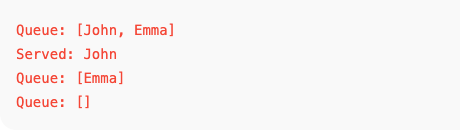
}

void displayQueue() {

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

}

}



// 6. Car Wash Queue

class CarWash {

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

void addNormal(String car) {

cars.addLast(car);

}

void addVIP(String car) {

cars.addFirst(car);

}

void washCar() {

if (!cars.isEmpty()) System.out.println("Washed: " + cars.poll());

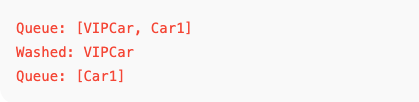
}

void displayQueue() {

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

}

}



// 7. Library Book Stack

class BookStack {

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

void addBook(String book) {

stack.push(book);

}

void removeBook() {

if (!stack.isEmpty()) System.out.println("Removed: " + stack.pop());

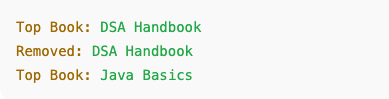
}

void peekBook() {

if (!stack.isEmpty()) System.out.println("Top Book: " + stack.peek());

}

}



// 8. Expression Evaluator

class ExpressionEvaluator {

int evaluatePostfix(String expr) {

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

for (char c : expr.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();

}

}

