Scenario based Assignment_1 - Punith Kumar

1. Browser History (Using Stack)

```
import java.util.*;

class ChronoNav {
    Stack<String> chronoBack = new Stack<>();
    Stack<String> chronoForward = new Stack<>();

    void accessPage(String url) {
        chronoBack.push(url);
        chronoForward.clear();
        System.out.println("Visited: " + url);
    }

    void goBack() {
        if (chronoBack.size() > 1) {
            chronoForward.push(chronoBack.pop());
            System.out.println("Back to: " + chronoBack.peek());
        } else {
            System.out.println("No previous page.");
        }
    }

    void goForward() {
        if (!chronoForward.isEmpty()) {
            String again = chronoForward.pop();
            chronoBack.push(again);
            System.out.println("Forward to: " + again);
        } else {
            System.out.println("No forward history.");
        }
    }
}
```

2. Print Queue (Using LinkedList as Queue)

```
import java.util.*;

class SpoolTower {
    Queue<String> docTrail = new LinkedList<>();

    void queueJob(String fileName) {
        docTrail.offer(fileName);
        System.out.println("Job added: " + fileName);
    }

    void printJob() {
        if (!docTrail.isEmpty()) {
            System.out.println("Printing: " + docTrail.poll());
        } else {
            System.out.println("Queue empty.");
        }
    }

    void viewQueue() {
        System.out.println("Pending Jobs: " + docTrail);
    }
}
```

3. Hospital Bed Management (Using LinkedList)

```
import java.util.*;

class MediWing {
    LinkedList<String> bedTracker = new LinkedList<>();

    void admitPatient(String patientID) {
        bedTracker.add(patientID);
        System.out.println("Admitted: " + patientID);
    }

    void releasePatient(String patientID) {
        if (bedTracker.remove(patientID)) {
            System.out.println("Discharged: " + patientID);
        } else {
            System.out.println("Patient not found.");
        }
    }

    void displayBeds() {
        System.out.println("Occupied Beds: " + bedTracker);
    }
}
```

4. Undo-Redo Function (Using Stack)

```
import java.util.*;

class DocManipulator {
    Stack<String> pastActions = new Stack<>();
    Stack<String> redoStack = new Stack<>();

    void doAction(String act) {
        pastActions.push(act);
        redoStack.clear();
        System.out.println("Action performed: " + act);
    }

    void undo() {
        if (!pastActions.isEmpty()) {
            String last = pastActions.pop();
            redoStack.push(last);
            System.out.println("Undo: " + last);
        } else {
            System.out.println("Nothing to undo.");
        }
    }

    void redo() {
        if (!redoStack.isEmpty()) {
            String redo = redoStack.pop();
            pastActions.push(redo);
            System.out.println("Redo: " + redo);
        } else {
            System.out.println("Nothing to redo.");
        }
    }
}
```

5. Ticket Booking System (Using Queue)

```
import java.util.*;

class TokenBoard {
    Queue<String> waitingLine = new LinkedList<>();

    void addPerson(String name) {
        waitingLine.offer(name);
        System.out.println(name + " added to queue.");
    }

    void serveNext() {
        if (!waitingLine.isEmpty()) {
            System.out.println("Serving: " + waitingLine.poll());
        } else {
            System.out.println("No one in queue.");
        }
    }

    void cancelTicket(String name) {
        if (waitingLine.remove(name)) {
            System.out.println("Cancelled: " + name);
        } else {
            System.out.println("Person not found.");
        }
    }
}
```

6. Car Wash Service Queue

```
import java.util.*;
```

```
class AutoFoam {
    LinkedList<String> washQueue = new LinkedList<>();

    void enqueueRegular(String car) {
        washQueue.addLast(car);
        System.out.println("Regular car: " + car);
    }

    void enqueueVIP(String car) {
        washQueue.addFirst(car);
        System.out.println("VIP car: " + car);
    }

    void washNext() {
        if (!washQueue.isEmpty()) {
            System.out.println("Washing: " + washQueue.poll());
        } else {
            System.out.println("No cars waiting.");
        }
    }
}
```

7. Library Book Stack (Using Stack)

```
import java.util.*;

class TomeShelf {
    Stack<String> bookVault = new Stack<>();

    void addBook(String title) {
        bookVault.push(title);
        System.out.println("Book added: " + title);
    }

    void removeBook() {
        if (!bookVault.isEmpty()) {
            System.out.println("Removed: " + bookVault.pop());
        } else {
            System.out.println("Shelf is empty.");
        }
    }

    void viewTopBook() {
        if (!bookVault.isEmpty()) {
            System.out.println("Top Book: " + bookVault.peek());
        } else {
            System.out.println("Shelf is empty.");
        }
    }
}
```

8. Expression Evaluator (Infix to Postfix & Evaluate)

```
import java.util.*;

class ExprOracle {

    int prec(char op) {
        return switch (op) {
            case '+', '-' -> 1;
            case '*', '/' -> 2;
            default -> -1;
        };
    }

    String convertToPostfix(String infix) {
        StringBuilder result = new StringBuilder();
        Stack<Character> opHold = new Stack<>();

        for (char ch : infix.toCharArray()) {
            if (Character.isLetterOrDigit(ch)) {
                result.append(ch);
            } else if (ch == '(') {
                opHold.push(ch);
            } else if (ch == ')') {
                while (!opHold.isEmpty() && opHold.peek() != '(')
                    result.append(opHold.pop());
                opHold.pop();
            } else {
                while (!opHold.isEmpty() && prec(ch) <= prec(opHold.peek()))
                    result.append(opHold.pop());
                opHold.push(ch);
            }
        }

        while (!opHold.isEmpty()) result.append(opHold.pop());
        return result.toString();
    }

    int evaluatePostfix(String postfix) {
        Stack<Integer> calc = new Stack<>();
        for (char ch : postfix.toCharArray()) {
            if (Character.isDigit(ch)) {
                calc.push(ch - '0');
            } else {
                int b = calc.pop();
                int a = calc.pop();
                switch (ch) {
                    case '+' -> calc.push(a + b);
                    case '-' -> calc.push(a - b);
                    case '*' -> calc.push(a * b);
                    case '/' -> calc.push(a / b);
                }
            }
        }
        return calc.pop();
    }
}
```

9. Reverse Queue Using Stack

```
import java.util.*;

class QueueFlipper {
    Queue<Integer> q = new LinkedList<>();

    void fillQueue(int[] nums) {
        for (int n : nums) q.offer(n);
    }

    void flipQueue() {
        Stack<Integer> temp = new Stack<>();
        while (!q.isEmpty()) temp.push(q.poll());
        while (!temp.isEmpty()) q.offer(temp.pop());
    }

    void showQueue() {
        System.out.println("Queue: " + q);
    }
}
```

10. Student Admission Queue with Emergency Slot

```
import java.util.*;

class ScholarGate {
    LinkedList<String> admissionLine = new LinkedList<>();

    void regularEntry(String name) {
        admissionLine.addLast(name);
        System.out.println(name + " added normally.");
    }

    void vipEntry(String name) {
        admissionLine.addFirst(name);
        System.out.println(name + " added as VIP.");
    }

    void admitNext() {
        if (!admissionLine.isEmpty()) {
            System.out.println("Admitted: " + admissionLine.poll());
        } else {
            System.out.println("No students waiting.");
        }
    }
```