INSTRUCTIONS :

1. Complete all questions in your designated project group.
2. All members must contribute to writing the codes. (i.e. 1 question = 1 person, and share the workload if there's an additional question relative to the actual number of members in your team (i.e. 5)). Ensure that all members must understand and explain codes from any of the questions.
3. During viva, all students in each team will be randomly asked to describe, answer and edit any of the answers provided. Marks will be given to your ability to present the answers.

Lab Report

Prepare a report to solve the above problems. The report should contain all the sections as below for each question:

Section	Description
1. Problem	Description on the problem
2. Solution	Explanation on how to solve the problems below
3. Sample Input & Output	A few sets of input and output (snapshot)
4. Source code	Java source code

Requirements

1. Group Assignment (Grouping is the same as your project group)
2. Cover page that includes all student matric number and full name.
3. Font: Times New Roman 12, Line Spacing: 1 ½ Spacing
4. Submit to Spectrum according to your OCC. **Deadline** : Before your viva session (W6).

Question 1 : Digital Root Calculator

Problem Statement

The digital root of a positive integer is the iterative sum of its digits until a single digit remains.

For example, 493193 → 4+9+3+1+9+3=29 → 2+9=11 → 1+1=2 → digital root = 2.

Sample output:

```
Enter number: 493193
Digital root: 2
```

Question 2: Balanced Parentheses Checker

Problem Statement:
Check whether a given string of parentheses is balanced.

Implement a method boolean isBalanced(String s) that returns true if every opening parenthesis has a matching closing one.

The string may contain other characters, which should be ignored.Only (and) count for checking. Print “Balanced” or “Not balanced” accordingly

Sample Output:

```
Enter expression: (3+5)*(7-(2+1))
Balanced
```

```
Enter expression: (5+6*(7-3)
Not balanced
```

Question 3: Lucky Ticket Verifier

Problem statement:

A ticket number is lucky if the sum of its first half of digits equals the sum of its second half.

For example, 123321 → (1+2+3) == (3+2+1).

Implement boolean isLuckyTicket(String ticket) where ticket is a string of even length

and digits only.

If not all characters are digits or length is odd, print “Invalid ticket number.”

Otherwise, check and print whether it is “Lucky” or “Unlucky.”

Sample output:

```
Enter ticket number: 124A23

Invalid ticket number.
```

```
Enter ticket number: 152125

Lucky
```

```
Enter ticket number: 133532

Unlucky
```

Question 4: Tic-Tac-Toe Winner Checker

Problem Statement:
Write a Java program that determines whether there is a winner in a completed 3 × 3 Tic-Tac-Toe board.

The program reads three input lines, each containing exactly three characters.

Valid characters are:
'X'– player X
'O'– player O
‘.’– empty square

Implement the following methods:

- char checkWinner(char[][] board)
  - Returns 'X' if player X has won,'O' if player O has won, or '.' if there is no winner.
  - A win occurs when the same symbol appears in any complete row, column, or one of the two diagonals.

- int countMoves(char[][] board, char player)
  - Returns the total number of moves made by the specified player.
  - This method can be used to validate or display statistics if desired.

In the main method:
1. Read the three board rows from the user.
2. Construct a 2D array of characters.
3. Call checkWinner() to determine the result.
4. Display the output as:

   a. "Winner: X" or "Winner: O" if a player has won,
   b. "No winner" if the game has no winning line.

Hints:
The method countMoves(char[][] board, char player) serves as a utility function to count how many times a specific player symbol ('X' or 'O') appears on the board. This method can be helpful for validation purposes. In a valid Tic-Tac-Toe game:Player X always starts first. Therefore, the number of X’s should be equal to or one more than the number of O’s.
If this condition is violated, the input board may be invalid.

Sample Output:

```
Enter row 1: XO.
Enter row 2: OX.
Enter row 3: ..O
Winner: O
```

```
Enter row 1: XXX
Enter row 2: O.O
Enter row 3: XOX

Invalid board: number of moves is not valid.
```

Question 5: Run-Length Encoding (RLE) Compressor/Decompressor

Problem Statement:

Run-Length Encoding (RLE) is a simple text-compression technique that replaces sequences of identical characters with a count followed by the character itself. Your task is to implement both compression and decompression modes.

The program must first read the mode of operation:

'C'– Compress

'D'– Decompress

Then it reads a line of text.

Implement the following methods:

1. String compress(String s)

   i. For each run of identical consecutive characters, replace it with <count><char>.

   ii. Example: AAABCCDDDD ⟶ 3A1B2C4D.

   iii. Spaces are also compressed; e.g., two spaces become 2 .

2. String decompress(String s)

i. Expands a valid RLE string back to its original text.

ii. Each run is a positive integer immediately followed by a character.

iii. Example: 3A1B2C4D → AAABCCDDDD.

iv. If the encoded input is invalid (for example, numbers not followed by a character),

print "Invalid encoding.".

3. Output format:

- For successful operations: Result: <decoded_or_encoded_text>
- For invalid decompression input: Invalid encoding.

Sample Output:

```
Mode (C/D): C

Text: A BBB

Result: 1A2 3B
```

```
Mode (C/D): D

Text: 1A2 3B

Result: A BBB
```

```
Mode (C/D): D

Text: 12

Invalid encoding.
```

Question 6 : Number Pattern Mirror Puzzle

Problem Statement:

Write a program that checks whether two integer arrays form a mirror pattern.

Two arrays A and B are mirror patterns if the first element of A equals the last of B,

the second of A equals the second last of B, and so on.

- Implement boolean isMirror(int[] a, int[] b) that returns true if both arrays are mirror patterns of each other.
- Both arrays must have the same length (1–50).
- Read two lines of comma-separated integers, convert them into arrays, and check if they are mirror

Sample Output:

```
Array A: 1, 2, 3
Array B: 3, 1, 2
Mirror pattern: false
```

```
Array A: 1, 2, 3, 4
Array B: 4, 3, 2, 1
Mirror pattern: true
```