Problem 1: Grade Calculator with Multiple Subjects

Concept: 1D Array, `for` loop, `if-else if-else` ladder, basic arithmetic.

Problem Statement: Write a Java program that takes the number of subjects and then the marks for each subject (out of 100) as input. Calculate the **average percentage** and assign a **grade** based on the following criteria:

- 90% - 100%: 'A'
- 75% - 89%: 'B'
- 60% - 74%: 'C'
- 40% - 59%: 'D'
- Below 40%: 'F'

Also, handle cases where marks entered are not between 0 and 100. If any mark is invalid, print an error and terminate.

Sample Input:

```
Number of subjects: 3
Marks for subject 1: 85
Marks for subject 2: 92
Marks for subject 3: 78
```

Sample Output:

```
Average Percentage: 85.0%
Grade: B
```

Sample Input 2:

```
Number of subjects: 2
Marks for subject 1: 105
```

Sample Output 2:

```
Error: Marks should be between 0 and 100.
```

Problem 2: Array Rotation

Concept: 1D Array, `for` loop, temporary array or shifting elements.

Problem Statement: Write a Java program that rotates a given 1D integer array to the **right** by a specified number of positions. The elements shifted off the end should re-enter from the beginning.

Sample Input:

```
Array: [1, 2, 3, 4, 5]
Positions to rotate: 2
```

Sample Output:

```
Rotated Array: [4, 5, 1, 2, 3]
```

Explanation for Sample Output:

- Initial: `[1, 2, 3, 4, 5]`
- Rotate 1 position right: `[5, 1, 2, 3, 4]`
- Rotate 2 positions right: `[4, 5, 1, 2, 3]`

Problem 3: Spiral Matrix Traversal

Concept: 2D Array, multiple nested `for` loops, boundary tracking, `while` loop.

Problem Statement: Write a Java program that traverses a given 2D integer matrix in a **spiral order** and prints its elements. You should start from the top-left corner, move right, then down, then left, then up, and continue spiraling inwards until all elements are visited.

Sample Input:

```
Matrix:
[[1,  2,  3,  4],
 [5,  6,  7,  8],
 [9, 10, 11, 12]]
```

Sample Output:

```
Spiral Traversal: 1 2 3 4 8 12 11 10 9 5 6 7
```

Problem 4: Saddle Point in a Matrix

Concept: 2D Array, nested `for` loops, finding minimums and maximums in rows/columns.

Problem Statement: Write a Java program to find a **"saddle point"** in a given 2D integer matrix. A saddle point is an element that is the **minimum in its row** and the **maximum in its column**. There might be zero, one, or multiple saddle points. If no saddle point exists, print a corresponding message.

Sample Input:

```
Matrix:
[[1, 2, 3],
 [4, 5, 6],
 [7, 8, 9]]
```

Sample Output:

```
No saddle point found.
```

Sample Input 2:

```
Matrix:
[[1, 2, 3],
 [4, 5, 6],
 [7, 1, 9]]
```

Sample Output 2:

```
Saddle point found at (2, 1) with value 1
```

(Row and column indices are 0-based)

Problem 5: Sudoku Validator

Concept: 2D Array, nested `for` loops, boolean arrays or hash sets for tracking, multiple validation checks.

Problem Statement: Write a Java program that checks if a given `9 X 9` integer grid represents a valid Sudoku solution. For a Sudoku solution to be valid, all of the following conditions must be met:

1. Each row must contain the digits 1-9 exactly once.
2. Each column must contain the digits 1-9 exactly once.
3. Each of the nine `3 X 3` sub-grids must contain the digits 1-9 exactly once.

Assume the input grid contains only digits from 1-9 or 0 (representing an empty cell, though for validation purposes, we'll assume filled cells for now as per the "solution" context).

Sample Input:

```
Sudoku Grid:
[[5, 3, 4, 6, 7, 8, 9, 1, 2],
 [6, 7, 2, 1, 9, 5, 3, 4, 8],
 [1, 9, 8, 3, 4, 2, 5, 6, 7],
 [8, 5, 9, 7, 6, 1, 4, 2, 3],
 [4, 2, 6, 8, 5, 3, 7, 9, 1],
 [7, 1, 3, 9, 2, 4, 8, 5, 6],
 [9, 6, 1, 5, 3, 7, 2, 8, 4],
 [2, 8, 7, 4, 1, 9, 6, 3, 5],
 [3, 4, 5, 2, 8, 6, 1, 7, 9]]
```

Sample Output:

```
Sudoku is Valid.
```

Sample Input 2:

```
Sudoku Grid:
[[5, 3, 4, 6, 7, 8, 9, 1, 2],
 [6, 7, 2, 1, 9, 5, 3, 4, 8],
 [1, 9, 8, 3, 4, 2, 5, 6, 7],
 [8, 5, 9, 7, 6, 1, 4, 2, 3],
 [4, 2, 6, 8, 5, 3, 7, 9, 1],
 [7, 1, 3, 9, 2, 4, 8, 5, 6],
 [9, 6, 1, 5, 3, 7, 2, 8, 4],
 [2, 8, 7, 4, 1, 9, 6, 3, 5],
 [3, 4, 5, 2, 8, 6, 1, 7, 1]] // '1' duplicated in last row
```

Sample Output 2:

```
Sudoku is Invalid.
```