The code for the solutions to the following problems should be written using Java. The code should be able to be directly executed from the command line interface or an IDE like Eclipse or IntelliJ Idea.

- You are required to submit a working sample of code that follows the best practices that you are aware of.
- The code should be easy to read and understand.
- The code should have proper comments and logging.
- Try and use OOPS Concepts in the code wherever applicable.

Along with the code, a write-up should also be provided explaining the logic used to solve the following problems.

1. Sudoku

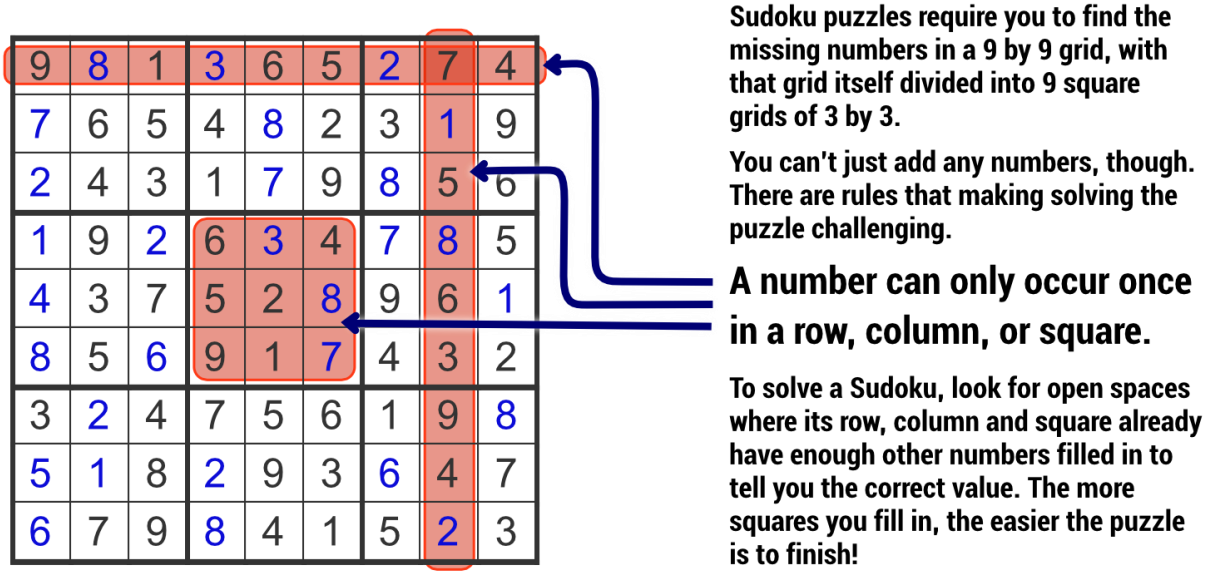
Description

Sudoku is a puzzle in which missing numbers are to be filled into a 9 by 9 grid of squares which are subdivided into 3 by 3 boxes so that every row, every column, and every box contains the numbers 1 through 9.

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

A Sudoku puzzle consists of 81 cells which are divided into nine columns, rows and regions. The task is now to place the numbers from 1 to 9 into the empty cells in such a way that in every row, column and 3×3 region each number appears only once.

A Sudoku has at least 17 given numbers but normally there are 22 to 30.



Problem Statement

The problem statement is to define a method which given a Sudoku puzzle, will try an enter a digit into the given row and column and check if the digit can be successfully inserted at the given location or not.

The method signature should be as follows:

```
public boolean checkValidity(int sudoku[][], int row, int column, int digit);
```

Here, **sudoku** is a 2-dimensional array of integers of size 9x9 which stores the given sudoku puzzle at each cell, eg: Sudoku[3][4] stores the digit at 4th row and 5th column of the Sudoku puzzle. **row** is an integer parameter which will denote the row where the digit is to be entered. **column** is an integer parameter which will denote the column where the digit is to be entered. **digit** represents the digit which is to be entered in the puzzle and whose validity is to be checked by the method.

The method will return **true** if the digit can be successfully inserted in the Sudoku puzzle with all the conditions of insertion satisfied. If any of the conditions is not able to met, the method should return **false**.

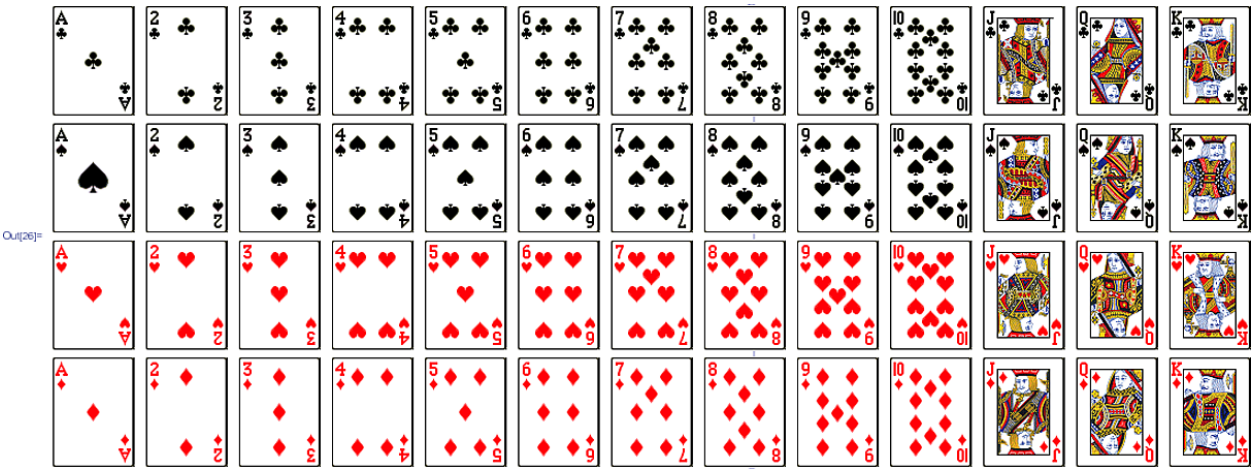
Example

checkValidity(mySudoku, 5, 7, 4) would check if the digit 4 can be inserted into the cell at the 6th row and 8th column in the given sudoku puzzle represented by the 2-dimensional array mySudoku. If all the rules are satisfied during insertion of the digit 4, the method will return true. If any of the rule is not able to be satisfied, the method will return false.

2. Playing Cards Sequence

Description

A standard deck of playing cards consists of 52 Cards in each of the four suits of Spades, Hearts, Diamonds, and Clubs. Each suit contains 13 cards: Ace, 2, 3, 4, 5, 6, 7, 8, 9, 10, Jack, Queen, King.



Problem Statement

For our problem, we will denote the four suits as follows:

- Spade will be represented by the character ‘S’
- Hearts will be represented by the character ‘H’
- Diamonds will be represented by the character ‘D’
- Clubs will be represented by the character ‘C’

We will denote each of the 13 cards as follows:

- Ace will be represented by the character ‘A’
- 2 will be represented by the character ‘2’
- 3 will be represented by the character ‘3’
- 4 will be represented by the character ‘4’
- 5 will be represented by the character ‘5’
- 6 will be represented by the character ‘6’
- 7 will be represented by the character ‘7’
- 8 will be represented by the character ‘8’
- 9 will be represented by the character ‘9’
- 10 will be represented by the String “10”
- Jack will be represented by the character ‘J’
- Queen will be represented by the character ‘Q’
- King will be represented by the character ‘K’

We will represent each card by it’s suit’s character, adding the character ‘#’ to it and then finally the card’s representation. Eg: An Ace of Spades will be represented as the String “S#A”, a 10 of Hearts will be represented by the String “H#10”, a 3 of Clubs will be represented as “C#3” and so on.

A sequence of cards is then defined as a set of 3 or more cards such that they are of the same suit and their values are in a sequential order. Eg: H#4, H#5, H#6 is a sequence. H#2, S#3, H#4 is not a sequence since the suits of the cards are not the same. C#7, C#8, C#10 is not a sequence as the face values are not in a sequential order.

To define the sequence, we will consider Ace as the value 1, Jack as the value 11, Queen as the value 12 and King as the value 13. Therefore, H#A,H#2,H#3 is a valid sequence. So is C#9,C#10,C#J,C#Q. S#J,S#Q,S#K is also a valid sequence. There are two exceptions to this definition as follows:

- Q,K,A is a valid sequence if the suits of the cards are the same.
- K,A,2 is not a valid sequence even if all the cards have the same suit.

The problem here then is, provided a string containing comma-separated representation of 3 or more cards, to find if the cards are in a sequence or not.

The method signature should be as follows:

```
public boolean checkSequence(String cards);
```

Here, **cards** is a string containing comma-separated representation of 3 or more cards. The method should return **true** if the cards are in a sequence as defined above. Otherwise, it should return **false**.

Example

If the input **cards** is defined as “H#A,H#2,H#3,H#4”, the method should return **true**

If the input **cards** is defined as “S#10,S#J,S#Q”, the method should return **true**

If the input **cards** is defined as “C#Q,C#K,C#A,C#2”, the method should return **false**

If the input **cards** is defined as “D#5,D#6,H#7”, the method should return **false**