Assignment 6

4x4 Tic-Tac-Toe Game 50 pts.

TOTAL: 50 pts.

General Requirements

- *Add comments to the source code you are writing:*
  - *Describe the purpose of every variable*
  - *Explain the algorithm you are using for solution*
  - *Add proper comments for all methods. Include @param, @return, and @throws tags*
- *Put all source (.java) and bytecode (.class) files into one folder and archive it using ZIP or RAR utility. Turn it in into the digital drop box.*

4x4 Tic-Tac-Toe Game

Write a program that allows two players to play a game of tic-tac-toe on 4x4 board.
Game user interface description:

1. Two players are playing a game, taking turns to make a move. Player 1 places Xs, and player 2 places Os on the 4x4 board. Player 1 wins when there are three Xs in a row on the game board (player 2 wins when there are three Os). Three Xs (or Os) can appear in a row, in a column, or diagonally across the board. A tie occurs when all of the locations on the board are full, but there is no winner.

2. On each turn the board is being displayed. There is a number of ways to organize better interface with the user. I would prefer to have a board that looks like this:

```
1    2    3    4
5    6    7    8
9    a    b    c
d    e    f    g
```

   When the user wants to make a move, he/she chooses a number (kind of a Hex system), that represents the placement of X or O on the board. Below is sample interaction with a player. Player's input appears in blue.

```
1    2    3    4
5    6    7    8
9    a    b    c
d    e    f    g
```

```
Player 1, make your move
=> 9

 1   2   3   4
 5   6   7   8
 X   a   b   c
 d   e   f   g

Player 2, make your move
=> a

 1   2   3   4
 5   6   7   8
 X   O   b   c
 d   e   f   g
```

3. This exchange goes on until one of the players wins or there is a tie.

Create a class called `TicTacToe` that will be handling user input/output and the logistics of the game. The class will have the following private fields:

1. `board` – a private two-dimensional array of size 4x4. The board will be storing characters. Initial look of the board is the following

1	2	3	4
5	6	7	8
9	a	b	c
d	e	f	g

   The numbering of the board intentionally starts with 1. Zero is avoided to prevent confusing it with O (a marker of a move).
2. `status` – stores game status. There are 2 possibilities here – either still playing or done (game came to an end, someone won)
3. `winner` – stores the info about winning/losing the game. There are 3 possible states here: X won, O won, tie
4. `whoseTurn` –stores who's turn it is to play now

The methods are the following

1. No-argument constructor that creates the initial setting on the board and gives initial values to all fields.
2. Public method `printBoard()` - prints the current board
3. Public method `input()` - provides the prompt and gathers input from the user. It must prompt the correct player, who's turn it is now to play, display the board using the `printBoard()` method, and gather input from the user. Input validation must be provided:

   a. User must provide a valid character that represents the cell on the board. The user must be asked to provide the character until the valid character has been entered.
   b. If the place that the user chose has been already taken (X or O already standing there), the method must notify the user and ask for another input.

   The method must also be taking care of changing the value of field that tracks whose turn it is to play (`whoseTurn`).

   Do not ignore the need to convert a digit/character into a position in the 2-dimentional array. This calls for another algorithm

4. Public method `simulateInput(char player, char letter)` – The method is needed for debugging purposes. It simulates a move of a user. The first parameter specifies the player (X or O), the second parameter specifies the move (as a character).
   The method must display the board using `printBoard()` method, and print out what player made a move and where. Data validation must be provided:
   a. If the character provided by parameter is invalid, an `IllegalArgumentException` with the message describing the issue must be thrown.
   b. If the player specified by parameter is playing out of turn, the `IllegalArgumentException` with the message describing the issue must be thrown.
   c. If the place on board specified by the parameter has been already taken (X or O already standing there), the `IllegalArgumentException` with the message describing the issue must be thrown.

   The method must also be taking care of changing the value of field that tracks whose turn it is to play (`whoseTurn`).
5. Public method `analyzeBoard()` – analyzes the current board and determines if there is a winning position present or if it is a tie. If the state of the game changes, the member variable that is tracking the game state must be changing value, and the info about the winner must be recorded in the field that stores that data.
6. Public method `done()` – accessor for `status` field. Method returns true if the game came to the end.
7. Public method `whoWon()` – returns a character (X, O or T – for tie), a value of the `winner` field

Create class named `TicTacToeSimulation.` The class must contain main().

1. Create an object of class `TicTacToe`
2. Call `printBoard()`, `simulateInput()`,`analyzeBoard()`to simulate the moves of two opponents. The purpose of this code is to TEST all the methods and make sure your `analyzeBoard()`algorithm works exactly right. Make sure to simulate the game multiple times to check on all the critical test cases.
   a. Use comments to explain why you chose the particular combination of moves, what part of the algorithm you are testing with the particular test case.

   b. Make sure to display the winner each time you think the game must come to an end.
3. Make sure to test `input()` method too. It should be a separate small test, not a part of the game simulation.

Create a class named `TicTacToeGame.` This class must contain main() and allow two users play the game using keyboard input.

1. Create an object of class `TicTacToe`
2. In a loop start playing the game, calling `printBoard()`, `input(),analyzeBoard()`until `done()` returns true.
3. After that display board for the last time and announce who is the winner.