

# Recitation 8 - Tic Tac Toe

A Tic Tac Toe board is composed of a 3 x 3 square board. Two players play the game, each having his/her/its game piece. The traditional game pieces are Xs and Os; however, they do not need to be. The game begins with player one choosing one of the nine squares. Once player one chooses, player one's game piece will occupy that square. Then player two selects one of the remaining squares and places his/her/its game piece in that square. The selection continues alternating between player one and player two until one of the following occurs:

1. One of the players has his/her/its game piece on the three horizontal or vertical squares or along the center diagonal squares. In that case, the player whose game pieces satisfy this condition is the winner.
2. All squares are occupied, and condition 1 is not satisfied. In that case, the game ends in a draw.

In this recitation exercise, you will implement a Tic Tac Toe board in a class called `TicTacTac`, which will be used in a class called `TicTacToeGame`. Since a game consists of game pieces, a class called `GamePiece` is already implemented. While this may seem like a useless class, you need only think about how this can be easily extended, for example, to implement on Greenfoot.

## The GamePiece Class

---

The `GamePiece` class is a simple class that uses characters as the basis of the game piece; however, it is straight forward to extend this to, for example, an Actor based game piece in Greenfoot. Take a look at the class and become familiar with its behavior.

## The TicTacToe Class

---

This is the implementation of the Tic Tac Toe board. You will be required to implement several of the methods in the `TicTacToe` class. The underlying array representing the board could have been implemented using a 1-dimensional array of length nine, or a 3 x 3 2-dimensional array. This recitation uses the latter representation.

The `TicTacToe` class declares a field called `player`, which is an array of two `GamePiece` type,

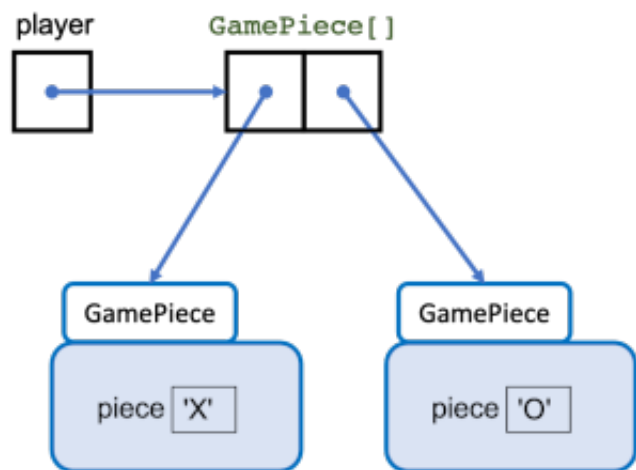
representing the game pieces of player one and two.

```
private GamePiece[] player;
```

and defined in the constructor as,

```
player = new GamePiece[2];  
player[0] = new GamePiece(p1);  
player[1] = new GamePiece(p2);
```

where p1 and p2 are game pieces. Diagrammatically, this may look like,



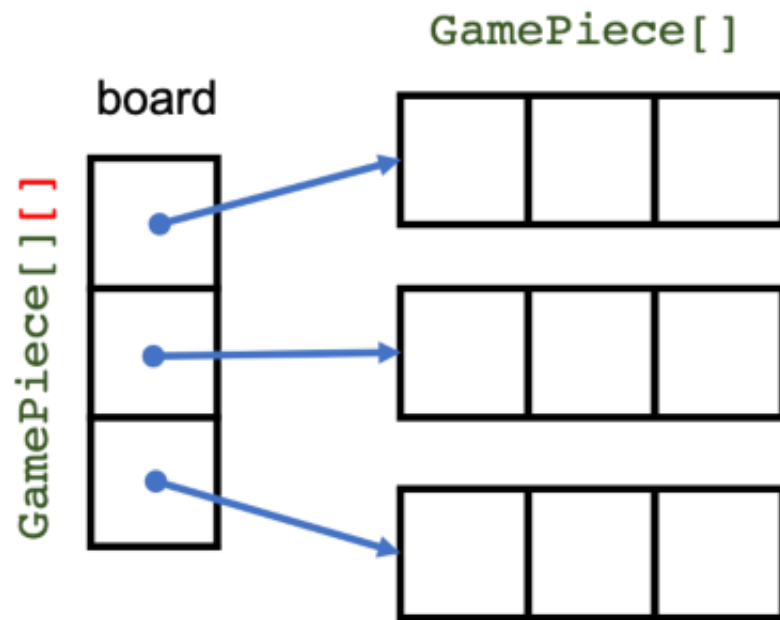
The TicTacToe class declares a field,

```
private GamePiece[][] board;
```

and defines the array in the constructor as,

```
board = new GamePiece[3][3];
```

Diagrammatically, this will look like,

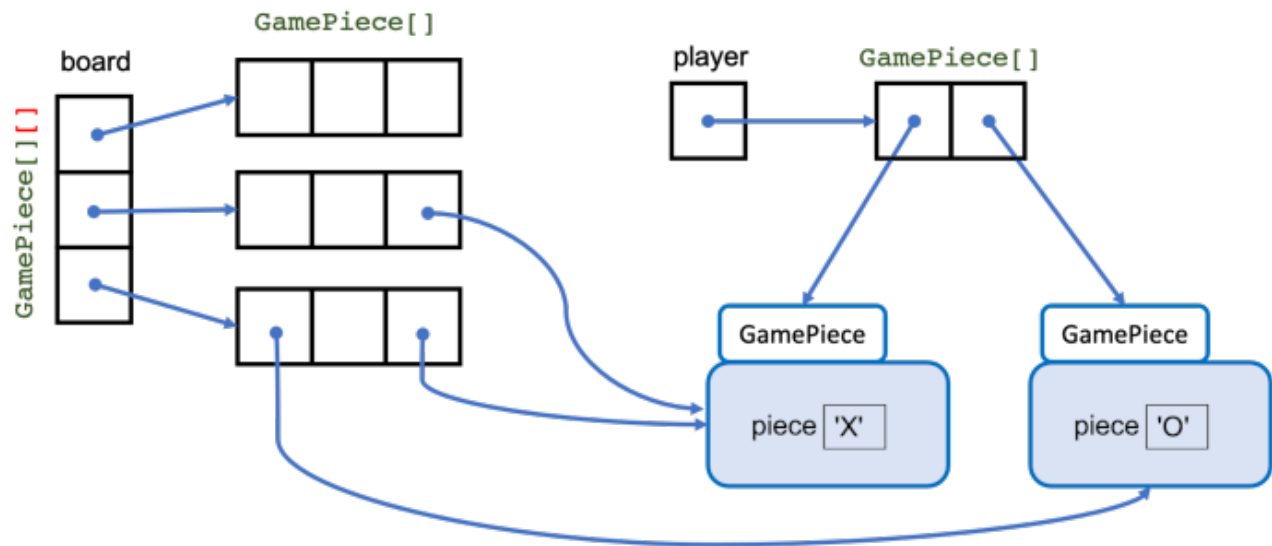


When a game piece occupies a board position, we will represent this by assigning a reference to one of the game pieces referred to in the `player` array.

So, a board that looks like the following,

		X
O		X

is represented at runtime in the following diagram.



## The Board

A string representation of TicTacToe object will be the board with the location of each square indicated or the game piece occupying that space. For example,

```
TicTacToe ttt = new TicTacToe(new GamePiece('X'),new GamePiece('O'));
System.out.print(ttt);
```

Since `System.out.print` requires a string to be printed, Java will automatically call `ttt.toString()` method since it exists. The `toString` method should return a string such that when printed looks like,

```
+---+---+---+
| 0 | 1 | 2 |
+---+---+---+
| 3 | 4 | 5 |
+---+---+---+
| 6 | 7 | 8 |
+---+---+---+
```

Let's assume that an X was added to location 0 and 4 and an O to location 8, then the board should look like,

```

+---+---+---+
| x | 1 | 2 |
+---+---+---+
| 3 | x | 5 |
+---+---+---+
| 6 | 7 | 0 |
+---+---+---+

```

## isEmpty() Method

This method returns `true` if the location is NOT occupied by a game piece (i.e. has a null for a value); false otherwise

Assuming `ttt` represents a TicTacToe object that looks like,

```

+---+---+---+
| x | 1 | 2 |
+---+---+---+
| 3 | x | 5 |
+---+---+---+
| 6 | 7 | 0 |
+---+---+---+

```

Then,

```
ttt.isEmpty(2);
```

should yield a `true`. However,

```
ttt.isEmpty(0);
```

should yield a `false`.

## isValid() Method

This method returns true if the location is an integer that represents one of the squares on the board ; false otherwise.

For example,

```
ttt.isValid(3);
```

should yield a `true` ; however,

```
ttt.isValid(9);
```

should yield aa `false` .

## **movesRemaining() Method**

This method should return the number of moves remaining on the board. When the game first starts,

```
int numOfMoves = ttt.movesRemaining();
```

should result in `numOfMoves` being 9.

Assuming `ttt` represents a TicTacToe object that looks like,

```
+---+---+---+
|  X  |  1  |  2  |
+---+---+---+
|  3  |  X  |  5  |
+---+---+---+
|  6  |  7  |  0  |
+---+---+---+
```

Then,

```
int numOfMoves = ttt.movesRemaining();
```

should result in `numOfMoves` being 6.

## **getPiece() Method**

This method returns the game piece at the provided location. This method makes a copy of the game piece at the provided location and returns a reference to the copy. This prevents data leak and possible corruption of the game piece.

## **clear() Method**

This method clears the board of all game pieces. For example, if `ttt` represents a TicTacToe object that looks like,

```

+---+---+---+
| x | 1 | 2 |
+---+---+---+
| 3 | x | 5 |
+---+---+---+
| 6 | 7 | 0 |
+---+---+---+

```

Then,

```
ttt.clear();
```

results in a board that looks like,

```

+---+---+---+
| 0 | 1 | 2 |
+---+---+---+
| 3 | 4 | 5 |
+---+---+---+
| 6 | 7 | 9 |
+---+---+---+

```

## getWinner() Method

This method returns a **copy** of the winner's GamePiece or null if there is no winner at the time the method is invoked. The method provides a 2-dimensional array called `combos` that contains the indices of all possible win configurations. Loop through that array and check if all the board location in each combo is filled with the same pieces. If yes, then return a copy of the game piece at that location; otherwise, return a null.

## toString

This method overrides the method of the superclass Object and returns a string that represents the current state of the board. For example, when a TicTacToe object is first created, printing it will result in the following:

```

TicTacToe ttt = new TicTacToe(new GamePiece('X'),new GamePiece('O'));
System.out.println(ttt);

```

```

+---+---+---+
| 0 | 1 | 2 |
+---+---+---+
| 3 | 4 | 5 |
+---+---+---+
| 6 | 7 | 9 |
+---+---+---+

```

At any location where there is a null, the integer location of the square is provided. If the following turns are played, then printing the board yields,

```

ttt.add(4); // player 1 starts - game piece is X
ttt.add(2); // player 2 plays next - game piece is O
System.out.println(ttt)

```

```

+---+---+---+
| 0 | 1 | 0 |
+---+---+---+
| 3 | X | 5 |
+---+---+---+
| 6 | 7 | 9 |
+---+---+---+

```

The `add()` method is implemented and converts the location to a (row,column) values and places the game piece at that location in the two-dimensional array that represents the board. It then updates who the next player is.

## Submission

---

You are required to implement the following methods in the TicTacToe class.

1. `isValid()`
2. `isEmpty()`
3. `movesRemaining()`
4. `getPiece()`
5. `clear()`
6. `getWinner()`

The `toString` method will be implemented in recitation. A unit test is provided on which to test the methods listed above.



# Recitation Activities

---

The `toString()` method and the `getWinner()` method will be implemented during recitation. You will also be given instructions on how to set up the run configuration to run a complete two-person game using the `TicTacToeGame` class..