Instance Variables :-

- **char board[][]** – To store the board.
- **char playerMark** – To store the current marker.
- **String playerName** – To store the current player name.
- **boolean markerIsPlaced** – To know if the marker is placed successfully

*Constructor has been used to instantiate the char board[][].

Methods/Function used :-

- **newGame()**
- **display()**
- **isFull()**
- **wins()**
- **checkRowCol(char mark1, char mark2, char mark3)**
  - **checkRowWins()**
  - **checkColWins()**
  - **checkDiagWins()**
- **setMark(char search)**
- **changeMark()**
- **changeName(String player1, String player2)**
- **playGame()**
- **wins()**
- **drawMatch()**

Methods Explanation

newGame() :-

Access type - Public
Return type - Void
Local variables – int c; int row; int col.

```
int c = '1';
for(int row = 0; row <= 2; row++) {
    for(int col = 0; col <= 2; col++) {
        board[row][col] = (char)c;
        c++;
    }
```

*We actually converted the ‘c’ variable to character type to make the code a bit optimized.

Traversing through every matrix of ‘board’ array and inputing the numbers from 1-9, variable ‘c’ has been used for this, and also incrementing the ‘c’ by one everytime after initializing to board array. We made each matrix of the array to be displayed as numbers for the ease of gameplay. Every time we call this method it will create a new board with new initialization to board.

we want the board to be look like this ->

1	2	3
4	5	6
7	8	9

display() :-

Access type - Public
Return type - Void
Local variables –int row; int col.

```
System.out.println("--------------------");
for(int row = 0; row <= 2; row++) {
    System.out.print("|  ");
    for(int col = 0; col <= 2; col++) {
        System.out.print(Character.toUpperCase(board[row][col]) + "  |
");
    }
    System.out.println();
}
System.out.println("--------------------");
```

This method is called for displaying the current state of board .
Example(this is a new board) :-

```
--------------------
| 1 | 2 | 3 |
| 4 | 5 | 6 |
| 7 | 8 | 9 |
--------------------
```

isFull() :-

Access type - Private
Return type - Boolean
Local variables – int c; int row; int col.

```
    int c ='1';
    for(int row = 0; row <= 2; row++) {
        for(int col = 0; col <= 2; col++) {
            if(board[row][col] == ((char)c))
                return false;
            c++;
        }
    }
    return true;
}
```

*We actually converted the 'c' variable to character type to make the code a bit optimized.

Here we traversed through each element of the board array and checked that if it's equal to the value of 'c' variable and we also increment the 'c' variable by 1 so that we can accurately check the board if it is full as the empty board (or unplayed board) actually have the initialization like this :-

1	2	3
4	5	6
7	8	9

If one of the element of the board is equal to the value in the 'c' variable then that means the board is not full; so, we return the loop with false.

checkRowColDiag(char mark1, char mark2, char mark3) :-

Access type - Private
Return type - Boolean
Local variables – char mark1; char mark1; char mark1.

```
private boolean checkRowColDiag(char mark1, char mark2, char mark3) {
     return ((mark1 == mark2) && (mark2 == mark3) && (mark1 == mark3));
}
```

This method is used to check if all the passed argument is equal with each other.
*We made this method to check the if the diagonal, column or row is equal.

---

checkColWins() :-

Access type - Private
Return type - Boolean
Local variables – int col.

```
    for(int col = 0; col < 3; col++) {
        if (checkRowColDiag(board[0][col], board[1][col], board[2][col])
== true) {
            return true;
        }
    }
    return false;
}
```

This method is used to iterate through every columns and if the one of the column have equal value then it's a win. We called the "checkRowColDiag(char mark1, char mark2, char mark3) " method to check if the values are equal.

---

checkRowWins() :-

Access type - Private
Return type - Boolean
Local variables –int row.

```
private boolean checkRowWins() {
    for(int row = 0; row < 3; row++) {
        if (checkRowColDiag(board[row][0], board[row][1], board[row][2])
== true)
            return true;
    }
    return false;
```

This method is used to iterate through every rows and if the one of the row have equal value then it's a win. We called the "checkRowColDiag(char mark1, char mark2, char mark3) " method to check if the values are equal.

---

checkDiagWins() :-

Access type - Private
Return type - Boolean
Local variables –int row; int col.

```
if(checkRowColDiag(board[0][0], board[1][1],board[2][2]) == true)
    return true;
else if(checkRowColDiag(board[0][2], board[1][1],board[2][0]) == true)
    return true;
else return false;
```

This method is used to check all diagonals and if the one of the diagonal have equal value then it's a win.

---

setMark(char search) :-

Access type - Private
Return type – Void.
Local variables – int row; int col.

```
markerIsPlaced = false;
for(int i = 0; i <= 2; i++) {
    for(int j = 0; j <= 2; j++) {
        if((char)board[i][j] == search) {
            board[i][j] = playerMark;
            markerIsPlaced = true;
            return;
        }
    }
}
System.out.println();
System.out.println("Marker is already used here");
System.out.println("Choose an Empty Space");
display();
```

Here, first we initialized the instance variable markerIsPlaced with false. Then, we take an argument (a matrix number of the board from the user); and by traversing through every element (or matrix) of the board, we check if that's equal to any value situated in board so that the marker will be placed on the empty and the exact matrix provided by the user. After getting true by checking if the provided value by user is equal to one the element in the board, we go ahead and change the current board element to the marker ('X' or 'O') and make the markerIsPlaced to true then return. If the loop complete it's iteration without having any true in the if block (which implies that the user input a wrong matrix/value or the user trying to place the marker on the used matrix) then it will display some messages and the current state of board with having the instance variable markerIsPlaced initialized false.

changeMark() :-

Access type - Private
Return type – Void.

```
if(playerMark == 'x') {
    playerMark = 'o';
}
else
    playerMark = 'x';
```

This method is called to swap between markers (playerMark). Here, we check, if the current marker is ‘X’ then change it to ‘O’; and if the current marker is ‘O’ then change it to ‘X’

---

changeName(String player1, String player2) :-

Access type - Private
Return type – Void.

```
if(playerName == player1)
    playerName = player2;
else
    playerName = player1;
```

This method is called to swap between player names (playerName). Here, we check, if the current player name is ‘player1’ then change it to ‘player2’; and if the current player name is ‘player2’ then change it to ‘player1’.

---

wins() :-

Access type - Private.
Return type – Boolean.

```
if(checkColWins() || checkRowWins() || checkDiagWins()) {
    return true;
}
else return false;
```

This method checks and return that any of the check win method from checkColWins() checkRowWins() checkDiagWins() is true or false

---

drawMatch() :-

Access type – Private.
Return type – Boolean.

```
return (isFull() && !wins());
```

This method checks and return that if the board is full (isFullI()) and there is no winner as well (wins() == false).

---

playGame() :-

Access type - Public
Return type – Void.
Local variables – String player1; String player2; char c.

```
newGame();
System.out.print("Enter Name (Player 1) ->  ");
String player1 = sc.nextLine();
System.out.print("Enter Name (Player 2) ->  ");
String player2 = sc.nextLine();
System.out.println(player1 + " Choose your marker ' x ' or ' o ' ");
playerMark = sc.next().charAt(0);
playerName = player1;
char c;
while(!wins() && !isFull()) {
    display();
    do {
        System.out.println("\"" +Character.toUpperCase(playerMark) + "\"
" + playerName + "'s Turn ");
        System.out.print("Place your mark (1-9) ->  "  );
        c = sc.next().charAt(0);
        setMark(c);
    } while(!markerIsPlaced);
    changeMark();
    changeName(player1,player2);
}
if(drawMatch()) {
    display();
    sc.nextLine();
    System.out.println("Draw Match");
}
else {
    display();
    changeMark();
    changeName(player1,player2);
    sc.nextLine();
    System.out.println("\"" + Character.toUpperCase(playerMark) + "\"  " +
playerName + "  Wins the Game ");
}
```

This is the gameplay method that is used to play the actual game. First we created a fresh new board with new initialization by calling the newGame() method. Then have two variables player1 and player2 to store the name of the players. We instantiate the playerMark with the user's choice ('X' or 'O').

Then we created a loop that will execute until there is no win or the board is not full (all the maker is not placed): while(!wins() && !isFull()), and we called the display() method to show the current state of board everytime the user successfully placed their marker on the matrix/position of the board. Inside this while loop we have another do-while loop that will place the marker in given position; it will execute until the marker is not placed (markerIsPlaced == false) and after placing the marker in given we also need to swap between next player and next marker, so, we called the changeMark() to swap between marker, and changeName(player1,player2), to swap between player.

Finally when the loop terminates we go ahead and check for the winner. We also should not forget that after the last marker was placed, the player as well as the marker was swapped. So, to get the actual winner we again have to swap the player and the marker. As you can see while displaying the winner we called the changeMark(), changeName(player1,player2) again to swap to the next player that actually won the game. We also have the drawMatch() method to know if the match is draw and display it.