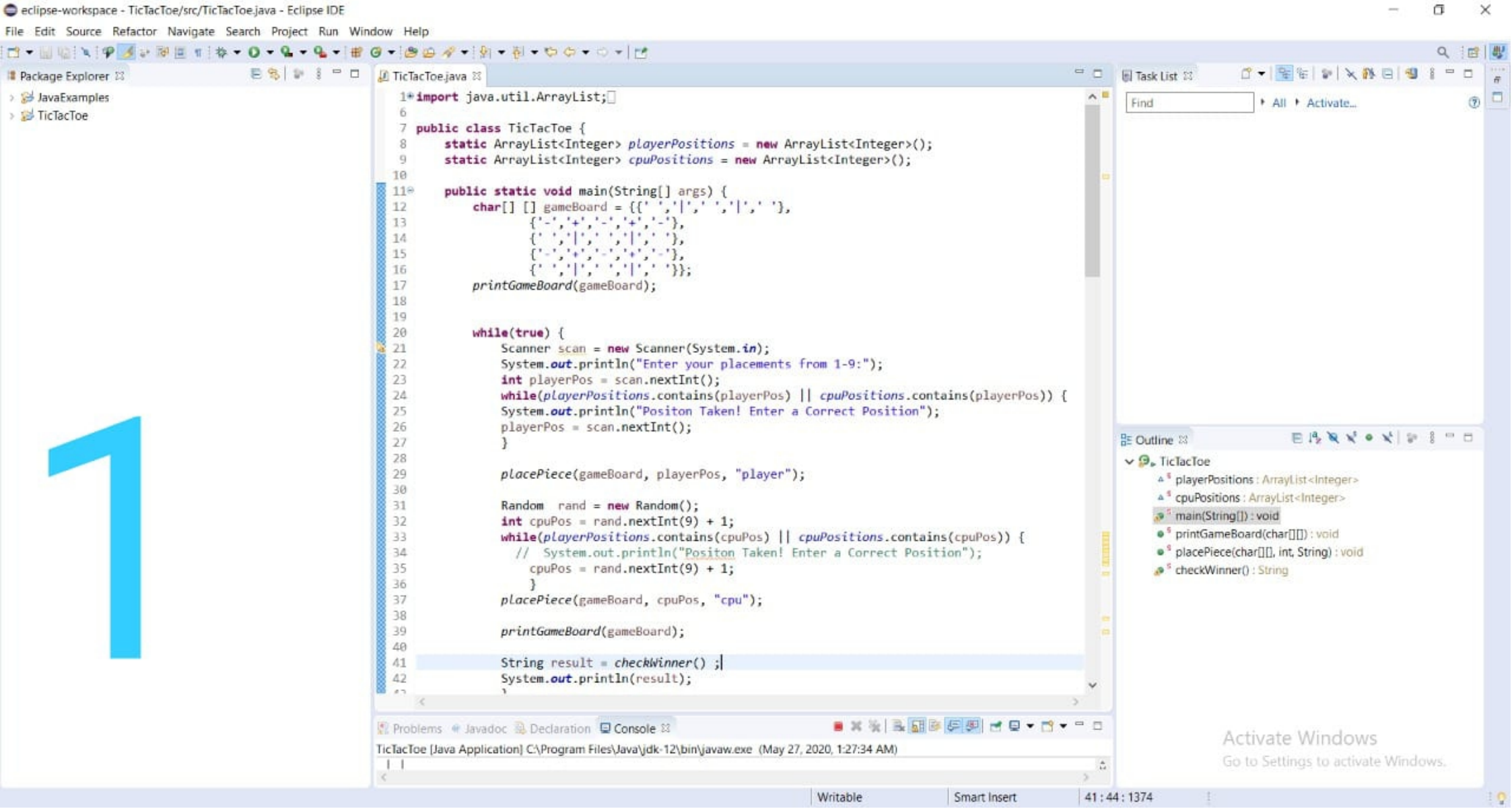
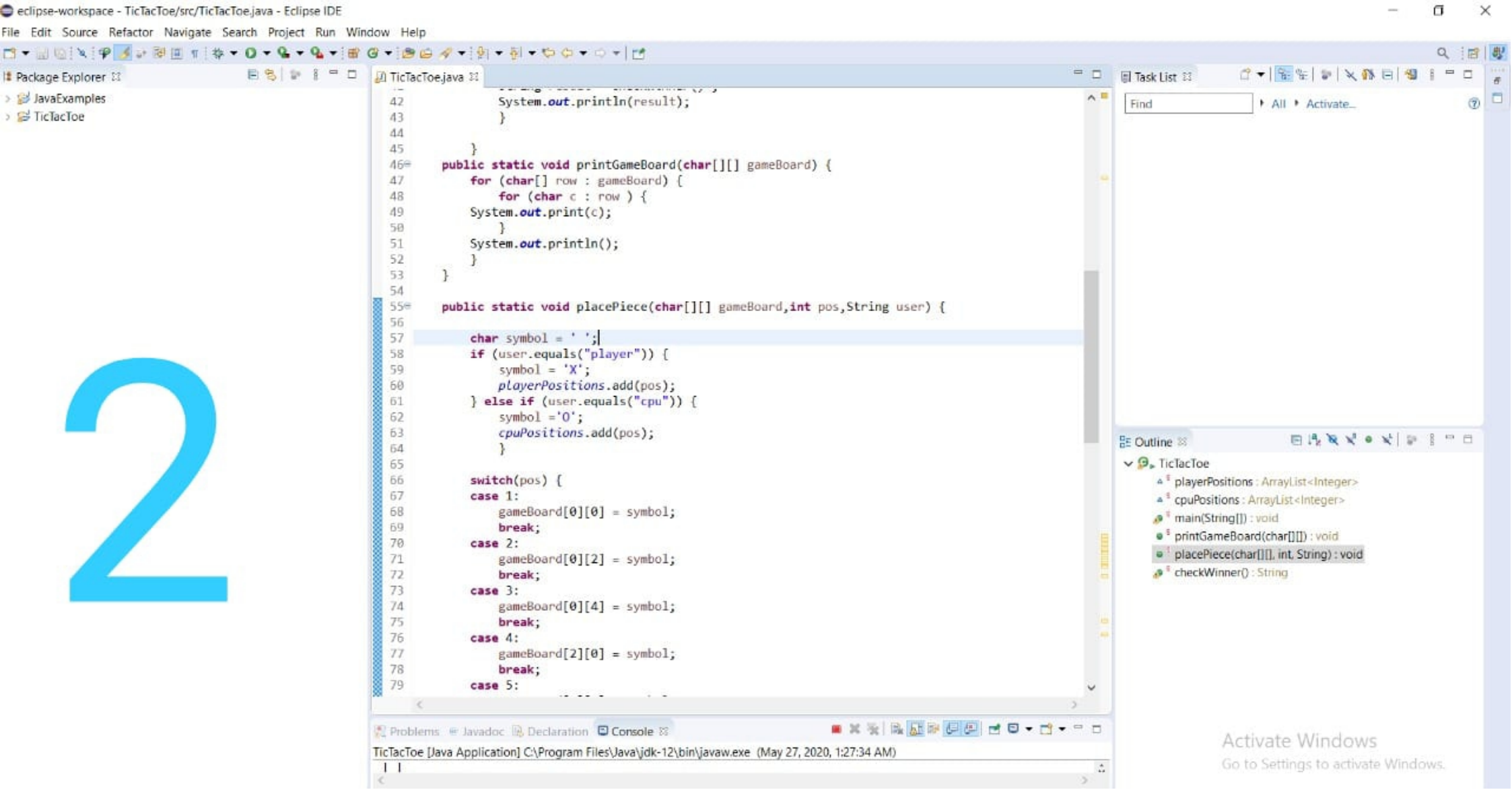
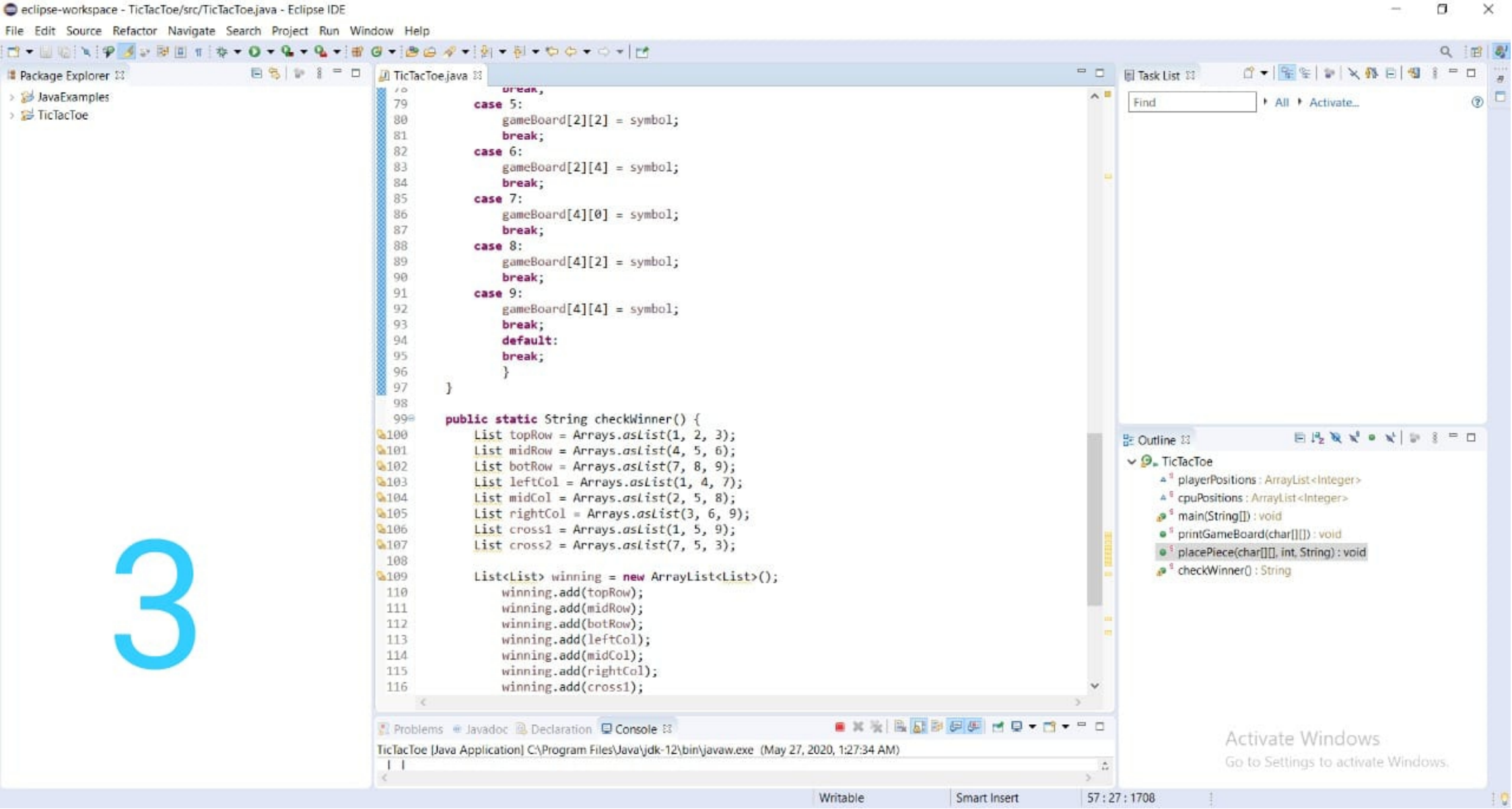
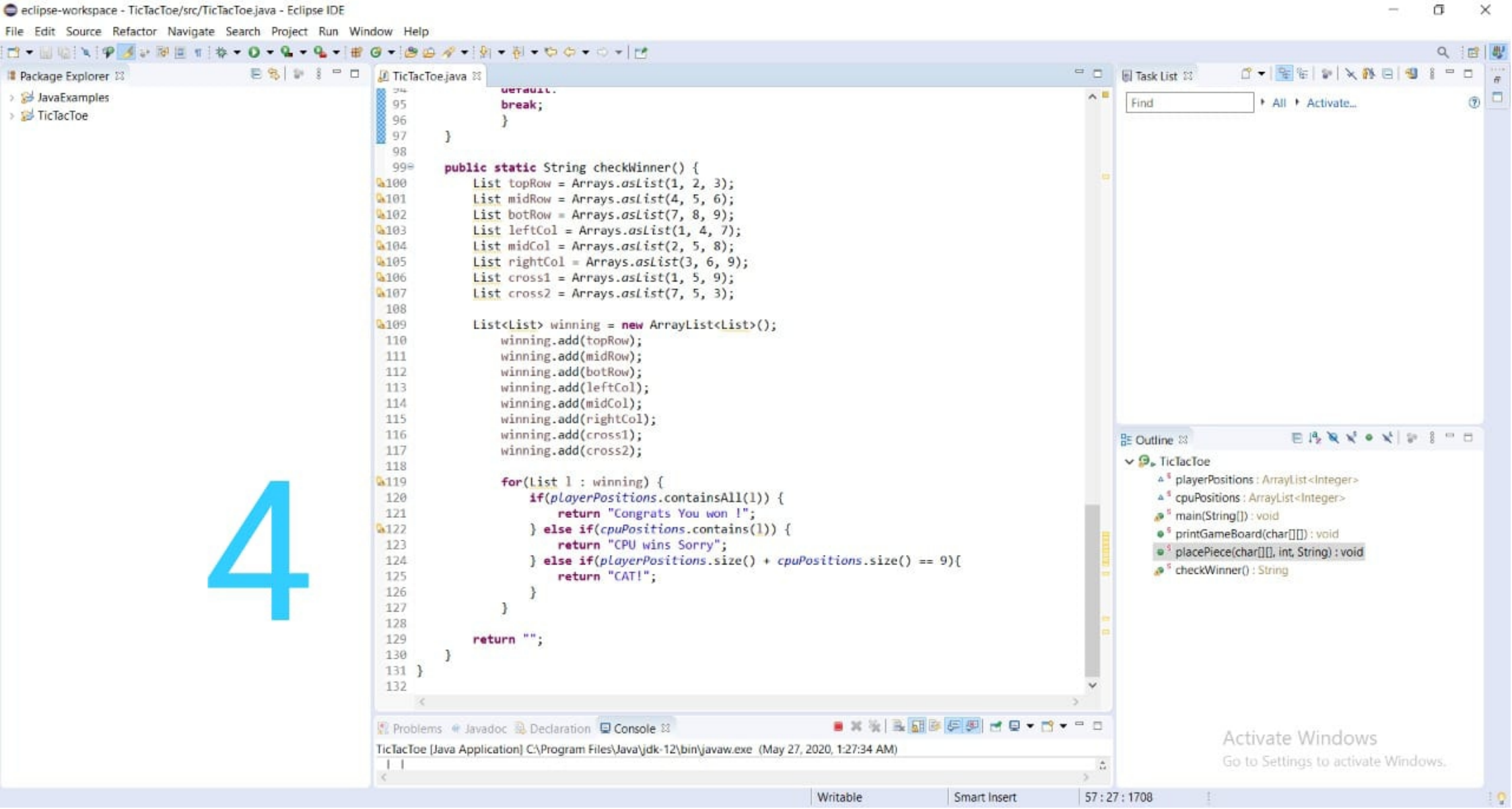

eclipse-workspace - TicTacToe/src/TicTacToe.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer
JavaExamples
TicTacToe
TicTacToe.java
1 import java.util.ArrayList;
6
7 public class TicTacToe {
8 static ArrayList<Integer> playerPositions = new ArrayList<Integer>();
9 static ArrayList<Integer> cpuPositions = new ArrayList<Integer>();
10
11 public static void main(String[] args) {
12 char[] [] gameBoard = {{' ','|',' ','|',' '},
13 {'-','+','-','+','-'},
14 {' ','|',' ','|',' '},
15 {'-','+','-','+','-'},
16 {' ','|',' ','|',' '}};
17 printGameBoard(gameBoard);
18
19
20 while(true) {
21 Scanner scan = new Scanner(System.in);
22 System.out.println("Enter your placements from 1-9:");
23 int playerPos = scan.nextInt();
24 while(playerPositions.contains(playerPos) || cpuPositions.contains(playerPos)) {
25 System.out.println("Positon Taken! Enter a Correct Position");
26 playerPos = scan.nextInt();
27 }
28
29 placePiece(gameBoard, playerPos, "player");
30
31 Random rand = new Random();
32 int cpuPos = rand.nextInt(9) + 1;
33 while(playerPositions.contains(cpuPos) || cpuPositions.contains(cpuPos)) {
34 // System.out.println("Positon Taken! Enter a Correct Position");
35 cpuPos = rand.nextInt(9) + 1;
36 }
37 placePiece(gameBoard, cpuPos, "cpu");
38
39 printGameBoard(gameBoard);
40
41 String result = checkWinner() ;
42 System.out.println(result);
Task List
Find
All
Activate...
Outline
TicTacToe
playerPositions : ArrayList<Integer>
cpuPositions : ArrayList<Integer>
main(String[]) : void
printGameBoard(char[][]) : void
placePiece(char[][], int, String) : void
checkWinner() : String
Problems
Javadoc
Declaration
Console
TicTacToe [Java Application] C:\Program Files\Java\jdk-12\bin\javaw.exe (May 27, 2020, 1:27:34 AM)
Activate Windows
Go to Settings to activate Windows.
Writable
Smart Insert
41 : 44 : 1374


eclipse-workspace - TicTacToe/src/TicTacToe.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer
JavaExamples
TicTacToe
TicTacToe.java
System.out.println(result);
}
}
public static void printGameBoard(char[][] gameBoard) {
for (char[] row : gameBoard) {
for (char c : row) {
System.out.print(c);
}
System.out.println();
}
}
public static void placePiece(char[][] gameBoard,int pos,String user) {
char symbol = ' ';
if (user.equals("player")) {
symbol = 'X';
playerPositions.add(pos);
} else if (user.equals("cpu")) {
symbol ='O';
cpuPositions.add(pos);
}
switch(pos) {
case 1:
gameBoard[0][0] = symbol;
break;
case 2:
gameBoard[0][2] = symbol;
break;
case 3:
gameBoard[0][4] = symbol;
break;
case 4:
gameBoard[2][0] = symbol;
break;
case 5:
Task List
Find
All
Activate...
Outline
TicTacToe
playerPositions : ArrayList<Integer>
cpuPositions : ArrayList<Integer>
main(String[]) : void
printGameBoard(char[][]) : void
placePiece(char[][], int, String) : void
checkWinner() : String
Problems
Javadoc
Declaration
Console
TicTacToe [Java Application] C:\Program Files\Java\jdk-12\bin\javaw.exe (May 27, 2020, 1:27:34 AM)
Activate Windows
Go to Settings to activate Windows.

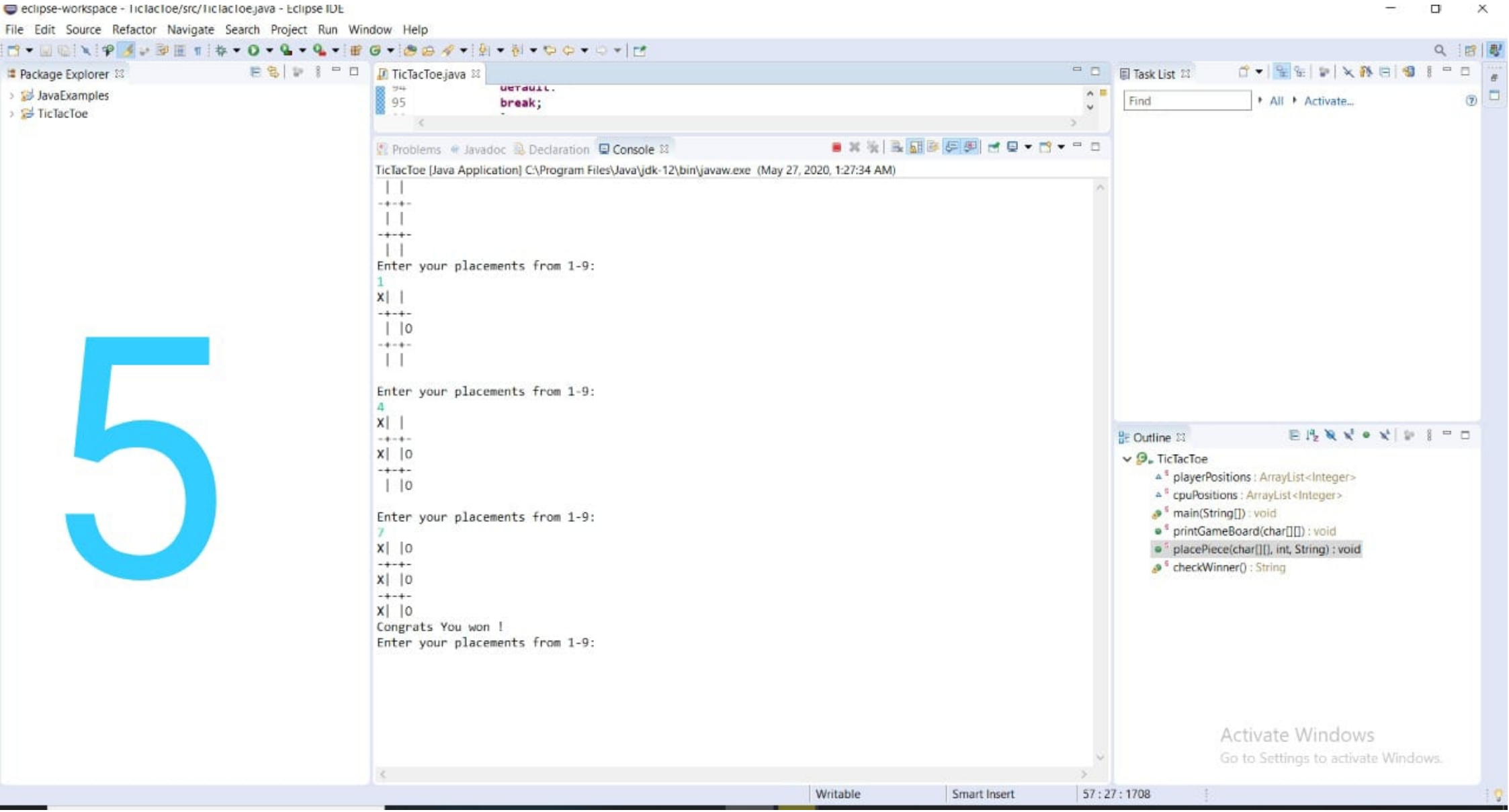
2


eclipse-workspace - TicTacToe/src/TicTacToe.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer
JavaExamples
TicTacToe
TicTacToe.java
79 case 5:
80 gameBoard[2][2] = symbol;
81 break;
82 case 6:
83 gameBoard[2][4] = symbol;
84 break;
85 case 7:
86 gameBoard[4][0] = symbol;
87 break;
88 case 8:
89 gameBoard[4][2] = symbol;
90 break;
91 case 9:
92 gameBoard[4][4] = symbol;
93 break;
94 default:
95 break;
96 }
97 }
98
99 public static String checkWinner() {
100 List topRow = Arrays.asList(1, 2, 3);
101 List midRow = Arrays.asList(4, 5, 6);
102 List botRow = Arrays.asList(7, 8, 9);
103 List leftCol = Arrays.asList(1, 4, 7);
104 List midCol = Arrays.asList(2, 5, 8);
105 List rightCol = Arrays.asList(3, 6, 9);
106 List cross1 = Arrays.asList(1, 5, 9);
107 List cross2 = Arrays.asList(7, 5, 3);
108
109 List<List> winning = new ArrayList<List>();
110 winning.add(topRow);
111 winning.add(midRow);
112 winning.add(botRow);
113 winning.add(leftCol);
114 winning.add(midCol);
115 winning.add(rightCol);
116 winning.add(cross1);
Task List
Find
All
Activate...
Outline
TicTacToe
playerPositions : ArrayList<Integer>
cpuPositions : ArrayList<Integer>
main(String[]) : void
printGameBoard(char[][]) : void
placePiece(char[][], int, String) : void
checkWinner() : String
Problems
Javadoc
Declaration
Console
TicTacToe [Java Application] C:\Program Files\Java\jdk-12\bin\javaw.exe (May 27, 2020, 1:27:34 AM)
Activate Windows
Go to Settings to activate Windows.
Writable
Smart Insert
57 : 27 : 1708

3


eclipse-workspace - TicTacToe/src/TicTacToe.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer
JavaExamples
TicTacToe
TicTacToe.java
break;
}
}
public static String checkWinner() {
List topRow = Arrays.asList(1, 2, 3);
List midRow = Arrays.asList(4, 5, 6);
List botRow = Arrays.asList(7, 8, 9);
List leftCol = Arrays.asList(1, 4, 7);
List midCol = Arrays.asList(2, 5, 8);
List rightCol = Arrays.asList(3, 6, 9);
List cross1 = Arrays.asList(1, 5, 9);
List cross2 = Arrays.asList(7, 5, 3);
List<List> winning = new ArrayList<List>();
winning.add(topRow);
winning.add(midRow);
winning.add(botRow);
winning.add(leftCol);
winning.add(midCol);
winning.add(rightCol);
winning.add(cross1);
winning.add(cross2);
for(List l : winning) {
if(playerPositions.containsAll(l)) {
return "Congrats You won !";
} else if(cpuPositions.contains(l)) {
return "CPU wins Sorry";
} else if(playerPositions.size() + cpuPositions.size() == 9){
return "CAT!";
}
}
return "";
}
}
Task List
Find
All
Activate...
Outline
TicTacToe
playerPositions : ArrayList<Integer>
cpuPositions : ArrayList<Integer>
main(String[]) : void
printGameBoard(char[][]) : void
placePiece(char[][], int, String) : void
checkWinner() : String
Problems
Javadoc
Declaration
Console
TicTacToe [Java Application] C:\Program Files\Java\jdk-12\bin\javaw.exe (May 27, 2020, 1:27:34 AM)
Activate Windows
Go to Settings to activate Windows.
Writable
Smart Insert
57 : 27 : 1708

4


eclipse-workspace - TicTacToe/src/TicTacToe.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer
JavaExamples
TicTacToe
TicTacToe.java
95 break;
Problems Javadoc Declaration Console
TicTacToe [Java Application] C:\Program Files\Java\jdk-12\bin\javaw.exe (May 27, 2020, 1:27:34 AM)
| |
-+-+-
| |
-+-+-
| |
Enter your placements from 1-9:
1
X| |
-+-+-
| |O
-+-+-
| |
Enter your placements from 1-9:
4
X| |
-+-+-
X| |O
-+-+-
| |O
Enter your placements from 1-9:
7
X| |O
-+-+-
X| |O
-+-+-
X| |O
Congrats You won !
Enter your placements from 1-9:
Task List
Find
All
Activate...
Outline
TicTacToe
playerPositions : ArrayList<Integer>
cpuPositions : ArrayList<Integer>
main(String[]) : void
printGameBoard(char[][]) : void
placePiece(char[][], int, String) : void
checkWinner() : String
Activate Windows
Go to Settings to activate Windows.
Writable
Smart Insert
57 : 27 : 1708

5

Introduction:

Tic-Tac-Toe is a very common game that is fairly easy to play. The rules of the game are simple and well-known. Because of these things, Tic-Tac-Toe is fairly easy to code up. In this tutorial, we will be looking at how to code a working game of Tic-Tac-Toe in Java. This tutorial assumes that you have knowledge of the basic syntax of Java, and access to a working Java compiler. This tutorial will use the Eclipse IDE.

General Outline:
There are many ways to implement a game of Tic-Tac-Toe in Java, so before we begin coding, we must think about how we will implement the game specifically. For this tutorial, we will be coding a text-based version of Tic-Tac-Toe. Our Tic-Tac-Toe will start out by printing the board, and then asking for input from the first player that will specify where on the board to place that player's mark. After placing the mark, we will print the board state again and then ask the other player for their move. That process will be continued until one player wins or the board is filled up (indicating that a tie occurred). The input that will be taken in to specify where to place a mark will be in the format of two integers, which specify the row and column where the mark is to be placed. Below is a sample of what a game will play like.

Step 1: Creating Your Project

The first step in coding anything is to make a new project! In your IDE, create a new Java project named TicTacToe. In this Instructable, the default package is what will be used. Inside of your project package, create two classes: Main.java, and TTT.java. Main.java will host the main method and will be used to run the code in TTT.java. TTT.java will contain a Tic-Tac-Toe object that contains the state of the board and methods to manipulate the game.

Step 2: Starting the TTT Class

Before we can make code to run our TTT object, we need to create a working TTT object. Since a TTT object is representative of one game of Tic-Tac-Toe, it must contain two member variables. Below are the member variables that should be placed into the TTT class followed by descriptions of why they are needed.

private char[][] board;

This variable is a 2D array of characters that will be representative of the three-by-three board of a game of Tic-Tac-Toe. It will hold the state of

and will be used to run the code in TTT.java. TTT.java will contain a Tic-Tac-Toe object that contains the state of the board and methods to manipulate the game.

Step 2: Starting the TTT Class

Before we can make code to run our TTT object, we need to create a working TTT object. Since a TTT object is representative of one game of Tic-Tac-Toe, it must contain two member variables. Below are the member variables that should be placed into the TTT class followed by descriptions of why they are needed.

private char[][] board;

This variable is a 2D array of characters that will be representative of the three-by-three board of a game of Tic-Tac-Toe. It will hold the state of the game inside of the TTT object at any given time.

private char currentPlayerMark;

This variable will hold either an 'x' or an 'o', representing which player's turn it is at any given point of time. The methods of the TTT class will use this when marking the board to determine which type of mark will be placed.

Step 3: Initializing Method Prototypes in the TTT Class

The following is a general setup of the program. Below are all of the method headers for the methods that belong inside of the TTT class. These methods will act upon the member variables of the TTT class to make the game of Tic-Tac-Toe mutable. Each one has a short description of what the method will do under it. These behaviours are necessary for playing a full game of Tic-Tac-Toe.

public TTT()

This is the constructor. It will be responsible for ensuring the board gets initialized properly, and for setting who the first player will be.

public void initializeBoard()

This method will initialize the board variable such that all slots are empty.

public void printBoard()

This method will print the Tic-Tac-Toe board to standard output.

methods will act upon the member variables of the TTT class to make the game of Tic-Tac-Toe mutable. Each one has a short description of what the method will do under it. These behaviours are necessary for playing a full game of Tic-Tac-Toe.

public TTT()

This is the constructor. It will be responsible for ensuring the board gets initialized properly, and for setting who the first player will be.

public void initializeBoard()

This method will initialize the board variable such that all slots are empty.

public void printBoard()

This method will print the Tic-Tac-Toe board to standard output.

public boolean isBoardFull()

This method will check whether or not the board is full. It will return true if the board is full and a false otherwise.

public boolean checkForWin()

This method will check to see if a player has won, and if so, it will return true.

private boolean checkRowsForWin()

This method will specifically check the rows for a win.

private boolean checkColumnsForWin()

This method will specifically check the columns for a win.

private boolean checkDiagonalsForWin()

This method will specifically check the diagonals for a win.

private boolean checkRowCol(char c1, char c2, char c3)

This method will check the three specified characters taken in to see if all three are the same 'x' or 'o' letter. If so, it will return true.

Note: If you code method stubs for all of the method headers into your TTT class, your compiler will likely inform you that your code has errors.

private boolean checkRowCol(char c1, char c2, char c3)

This method will check the three specified characters taken in to see if all three are the same 'x' or 'o' letter. If so, it will return true.

Note: If you code method stubs for all of the method headers into your TTT class, your compiler will likely inform you that your code has errors.

This is normal. The compiler is simply expecting for a value to be returned for all non-void methods.

Step 4: Initialize the Board

public void initializeBoard()

This will set the board to all empty values. Inside this method, you must create 2 for loops inside of each other that will loop through all the rows and columns setting each space to '-'. To loop through the rows, create a for loop and an integer, in this case named i, to represent which row we are currently observing.

```
for (int i = 0; i < 3; i++) { }
```

Inside of this for loop, we will create a second for loop with an integer j to represent which column we are currently observing.

```
for (int j = 0; j < 3; j++) { }
```

Inside of the second for loop, we set the board location to '-'. With both loops completed and nested properly, we can iterate through every place inside of the board 2D array.

```
board[i][j] = '-';
```

Attached to this step is an image showing one possible implementation of the initializeBoard() method.

Step 5: Printing the Board

The initial printed board will look like the first image .

board[i][j] = '-';

Attached to this step is an image showing one possible implementation of the initializeBoard() method.

Step 5: Printing the Board

The initial printed board will look like the first image .

It will be handled in the method public void printBoard(), which is located in the TTT class. To print the board we must access every place in the 2D array named board in our TTT class. Because we are dealing with a 2D array, this will be handled with nested for loops.

First, we just need to print a line of dashes (13 of them in this case) designating the top of the board. Below that, we need a for loop that will loop through each of the three rows. This loop will contain a call to print a '|' character, another for loop to loop through the columns, and a call to the System.out.println() function to print a new line and the next 13 dashes to the screen.

Our inner for loop will also only loop through three columns. Since our outer for loop already printed the first | character of each row of the board, we can go ahead to print the character that belongs in the box. To do this, we'll print the character at that row and column using board[i][j] (i being the variable used for the outer for loop, which was the row, and j being the variable used for the inner loop, which is the column.) This print statement will also contain a concatenated | character, to separate the boxes.

The only thing left is to print the last call to print the new line to separate each row, followed by the 13 dashes. The second attached image shows an example of what the described print function might look like.

Step 6: Checking for a Winner (Part 1)

There are three different functions to check for a win: rows, columns, and diagonals. Computers have to separate these into different conditions because they are all different in terms of arrays. checkForWin() will be our main function to test all 3 of these functions for each scenario that the user input has affected.

Step 6: Checking for a Winner (Part 1)

There are three different functions to check for a win: rows, columns, and diagonals. Computers have to separate these into different conditions because they are all different in terms of arrays. checkForWin() will be our main function to test all 3 of these functions for each scenario that the user input has affected.

For checkForWin() method: You simply need a return statement that calls upon the three different functions. If checking the rows for win does not return true then check the columns for win, etc. Inside of the return statement should look like: checkRowsForWin() || checkColumnsForWin() || checkDiagonalsForWin()

For checkRowsForWin() method: We are looping through the rows to see if there are any winners. This will require one for loop with an if statement inside of it. The for loop will be incrementing through integer i so as to check each row. for (int i = 0; i < 3; i++) The if statement compares each space in the row to each other and gives a 'true' value if they are all equal. For instance if the row had three x's in a row, the method would return true. if (checkRowCol(board[i][0], board[i][1], board[i][2]) == true) So, inside of this if statement, there should be a: return true; And after the for loop, if the method never stopped, then the method needs return that this row did not have three consecutive matching symbols. Therefore, right before closing off the method with its final '}', we will write: return false; Satisfying the need to return a boolean.

For checkColumnsForWin() method: Copy and paste the contents of the checkRowsForWin() method. The only change will be inside of the if statement. Instead of incrementing through the rows, we will be incrementing through the columns. So while in checkRowsForWin has an if statement that says: if (checkRowCol(board[i][0], board[i][1], board[i][2]) == true) checkColumnsForWin() will have an if statement that says: if (checkRowCol(board[0][i], board[1][i], board[2][i]) == true) Other than that, everything else in the method remains the same.

For checkDiagonalsForWin() method: Everything written can be contained inside the parentheses of a return() statement. The first check we will perform is on the diagonal from the top left corner to the bottom right corner. To do this, we check all the spaces that would be included in this section. checkRowCol(board[0][0], board[1][1], board[2][2]) == true)

board[i][2]) == true) checkColumnsForWin() will have an if statement that says: if (checkRowCol(board[0][i], board[1][i], board[2][i]) == true) Other than that, everything else in the method remains the same.

For checkDiagonalsForWin() method: Everything written can be contained inside the parentheses of a return() statement. The first check we will perform is on the diagonal from the top left corner to the bottom right corner. To do this, we check all the spaces that would be included in this section. checkRowCol(board[0][0], board[1][1], board[2][2]) == true)

Then we will have one more statement, but we will separate the two by an OR symbol: '||' The second statement will check from the top right corner to the bottom left corner. checkRowCol(board[0][2], board[1][1], board[2][0]) == true) So your final product of the checkDiagonalsForWin() method should be a return(), and inside it should contain the first statement OR the second statement.

Step 7: Check for a Winner (Part 2)

Now we have to make sure if a player gets three in a row, he or she wins. checkRowCol() will be a function that will compare all three letters to each other, and if they do match, then return true.

For checkRowCol() method: This method is used by the other methods to send down three values. We first check to make sure the first value is not an empty one such as '-'. Then we compare the first value to the second, and the second to the third, and if and only if all three values are the same AND they are not empty statements, then this method will return true. So inside of one return() statement, our first statement will check this is not a '-'. (c1 != '-') Separate the first and second statements with an '&&' The second statement will see if the first value is equal to the second value. (c1 == c2) Separate the second and third statements with an '&&' The third statement will see if the second value is equal to the third. (c2 == c3) So your final checkRowCol() method will be one return() containing the first statement && the second statement && the third statement.

Step 8: Changing Between Players (x and O)

public void changePlayer()

The changePlayer() method will swap the variable currentPlayerMark between x and o. To do this, just check what the variable currently holds. If the variable is holding an 'x', then change it to an 'o'. Otherwise, change it to an 'x'.

public boolean placeMark(int row, int col)

second statement will see if the first value is equal to the second value. (c1 == c2) Separate the second and third statements with an '&&' The third statement will see if the second value is equal to the third. (c2 == c3) So your final checkRowCol() method will be one return() containing the first statement && the second statement && the third statement.

Step 8: Changing Between Players (x and O)

public void changePlayer()

The changePlayer() method will swap the variable currentPlayerMark between x and o. To do this, just check what the variable currently holds. If the variable is holding an 'x', then change it to an 'o'. Otherwise, change it to an 'x'.

public boolean placeMark(int row, int col)

The placeMark() method will place the correct letter onto the specified row and col in the board variable (taken in as parameters). It will return true if it was a valid placement. Otherwise, no modification to the board variable will be made, and the player will have to try and place their letter on a different place, as an invalid spot was selected or a spot where a player already placed their letter was selected. To accomplish this behavior, a few things must be checked. First, ensure (using an if statement) that the row argument was between 0 and 2. Next, check to ensure that the col argument was between 0 and 2. Finally, check to make sure that the spot in question currently contains a '-', signifying that no payer has marked that spot yet. If all three of these conditions check out, then place a mark (the type of which is specified by the class variable currentPlayerMark) at the location specified by row and col and then return true. If any of the three conditions are not met, then noting should happen and false should be returned.

Attached to this step are images showing possible implementations of the methods mentioned above.

Step 9: Player Input and Playing the Game

Now that the TTT class and all of its methods are completed, a main method that runs through an entire game of Tic-Tac-Toe using out TTT object must be created. The main method must do quite a few things in order to run a full game of Tic-Tac-Toe.

First, it must create a Scanner object to take input from System.in. Also, it must instantiate a TTT object to play the Tic-Tac-Toe game with. After these things, it must initialize the board of the TTT object by calling it's initializeBoard() method.

methods mentioned above.

Step 9: Player Input and Playing the Game

Now that the TTT class and all of its methods are completed, a main method that runs through an entire game of Tic-Tac-Toe using out TTT object must be created. The main method must do quite a few things in order to run a full game of Tic-Tac-Toe.

First, it must create a Scanner object to take input from System.in. Also, it must instantiate a TTT object to play the Tic-Tac-Toe game with. After these things, it must initialize the board of the TTT object by calling it's initializeBoard() method.

After these steps are completed, actual game play must be accounted for. To go through turns, a do while loop is required. The loop should break out when the game is over, meaning whenever the TTT object's board is full or has a winner. Inside the loop, the current board state should be printed before each turn so show the player what spaces are available and what spaces are taken. Then, two inputs should be taken in signifying the row and column to place a mark for the turn. After this input is taken in, the mark should be placed using the TTT object's method, and the player should be changed as well using the TTT object's method.

Below the while loop that handles all turns until the end of the game, it will be necessary to declare who the winner of the game is (or if the game was a tie). To do this, check if the game was a tie first by checking if both the board was full and there was no winner. If these things are the case, then print out that the game was a tie. Otherwise, print out who won by printing out the opposite of the current state of the TTT object's currentPlayerMark variable. This can be accomplished by first calling the TTT object's changePlayer() method and then using the TTT object's getCurrentPlayerMark() method to get the state of the currentPlayerMark variable. It may also be nice to print the board out once more to show the final state of the board.

Step 10: Use Class, Compile, and Run

Once you have gotten to this point, your project should be complete enough to run. If there are errors or missing parentheses or semicolons, now would be the time to find these and fix them. If you are at all lost as to how your project should look, we have included a downloadable file for you to compare your code to a working final product. Go ahead and compile your project.