**Student Name: Chioma Ukaegbu**

**BOARD.JAVA**

/\*\*

\* The Board class represents the game board for tic-tac-toe.

\* It manages the state of the board, checks for game outcomes, and displays the board.

\*/

**public** **class** Board **implements** Constants {

**private** **char** theBoard[][];

**private** **int** markCount;

/\*\*

\* Constructs a new Board instance and initializes the game board.

\* The board is initially empty (filled with SPACE\_CHAR).

\*/

**public** Board() {

markCount = 0;

theBoard = **new** **char**[3][];

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

theBoard[i] = **new** **char**[3];

**for** (**int** j = 0; j < 3; j++)

theBoard[i][j] = ***SPACE\_CHAR***;

}

}

/\*\*

\* Gets the mark (X, O, or SPACE\_CHAR) at a specified row and column on the board.

\* **@param** row The row index.

\* **@param** col The column index.

\* **@return** The mark at the specified position.

\*/

**public** **char** getMark(**int** row, **int** col) {

**return** theBoard[row][col];

}

/\*\*

\* Checks if the game board is full (no empty spaces left).

\* **@return** true if the board is full, false otherwise.

\*/

**public** **boolean** isFull() {

**return** markCount == 9;

}

/\*\*

\* Checks if the 'X' player wins the game.

\* **@return** true if 'X' wins, false otherwise.

\*/

**public** **boolean** xWins() {

**if** (checkWinner(***LETTER\_X***) == 1)

**return** **true**;

**else**

**return** **false**;

}

/\*\*

\* Checks if the 'O' player wins the game.

\* **@return** true if 'O' wins, false otherwise.

\*/

**public** **boolean** oWins() {

**if** (checkWinner(***LETTER\_O***) == 1)

**return** **true**;

**else**

**return** **false**;

}

/\*\*

\* Displays the current state of the game board including row and column headers.

\*/

**public** **void** display() {

displayColumnHeaders();

addHyphens();

**for** (**int** row = 0; row < 3; row++) {

addSpaces();

System.***out***.print(" row " + row + ' ');

**for** (**int** col = 0; col < 3; col++)

System.***out***.print("| " + getMark(row, col) + " ");

System.***out***.println("|");

addSpaces();

addHyphens();

}

}

/\*\*

\* Places a mark (X or O) on the game board at the specified row and column.

\* **@param** row The row where the mark is placed.

\* **@param** col The column where the mark is placed.

\* **@param** mark The mark to be placed (X or O).

\*/

**public** **void** addMark(**int** row, **int** col, **char** mark) {

theBoard[row][col] = mark;

markCount++;

}

/\*\*

\* Clears the game board by resetting all cells to SPACE\_CHAR and mark count to 0.

\*/

**public** **void** clear() {

**for** (**int** i = 0; i < 3; i++)

**for** (**int** j = 0; j < 3; j++)

theBoard[i][j] = ***SPACE\_CHAR***;

markCount = 0;

}

/\*\*

\* Checks if a player with a specified mark wins the game. Checks for a winning pattern horizontally, vertically or diagonally

\* **@param** mark The mark to check for (X or O).

\* **@return** 1 if the player wins, 0 if not.

\*/

**int** checkWinner(**char** mark) {

**int** row, col;

**int** result = 0;

**for** (row = 0; result == 0 && row < 3; row++) {

**int** row\_result = 1;

**for** (col = 0; row\_result == 1 && col < 3; col++)

**if** (theBoard[row][col] != mark)

row\_result = 0;

**if** (row\_result != 0)

result = 1;

}

**for** (col = 0; result == 0 && col < 3; col++) {

**int** col\_result = 1;

**for** (row = 0; col\_result != 0 && row < 3; row++)

**if** (theBoard[row][col] != mark)

col\_result = 0;

**if** (col\_result != 0)

result = 1;

}

**if** (result == 0) {

**int** diag1Result = 1;

**for** (row = 0; diag1Result != 0 && row < 3; row++)

**if** (theBoard[row][row] != mark)

diag1Result = 0;

**if** (diag1Result != 0)

result = 1;

}

**if** (result == 0) {

**int** diag2Result = 1;

**for** (row = 0; diag2Result != 0 && row < 3; row++)

**if** (theBoard[row][3 - 1 - row] != mark)

diag2Result = 0;

**if** (diag2Result != 0)

result = 1;

}

**return** result;

}

/\*\*

\* Displays column headers for the game board.

\*/

**void** displayColumnHeaders() {

System.***out***.print(" ");

**for** (**int** j = 0; j < 3; j++)

System.***out***.print("|col " + j);

System.***out***.println();

}

/\*\*

\* Adds hyphens to visually separate rows on the game board.

\*/

**void** addHyphens() {

System.***out***.print(" ");

**for** (**int** j = 0; j < 3; j++)

System.***out***.print("+-----");

System.***out***.println("+");

}

/\*\*

\* Adds spaces to visually separate cells within rows on the game board.

\*/

**void** addSpaces() {

System.***out***.print(" ");

**for** (**int** j = 0; j < 3; j++)

System.***out***.print("| ");

System.***out***.println("|");

}

}

**GAME.JAVA**

**import** java.io.\*;

/\*\*

\* The `Game` class represents the main entry point for a tic-tac-toe game.

\* It initializes the game board, appoints a referee, and manages player setup.

\*/

**public** **class** Game **implements** Constants {

**private** Board theBoard; // Represents the game board.

**private** Referee theRef; // Manages the game and checks for game outcomes.

/\*\*

\* Constructs a new `Game` instance and initializes the game board.

\*/

**public** Game() {

theBoard = **new** Board();

}

/\*\*

\* Appoints a referee to manage the game and runs the game.

\*

\* **@param** r The referee responsible for controlling the game.

\* **@throws** IOException if there is an error during game setup.

\*/

**public** **void** appointReferee(Referee r) **throws** IOException {

theRef = r;

theRef.runTheGame();

}

/\*\*

\* The main method for starting and running a tic-tac-toe game.

\*

\* **@param** args Command-line arguments (not used).

\* **@throws** IOException if there is an error during input/output operations.

\*/

**public** **static** **void** main(String[] args) **throws** IOException {

Referee theRef;

Player xPlayer, oPlayer;

BufferedReader stdin;

Game theGame = **new** Game();

stdin = **new** BufferedReader(**new** InputStreamReader(System.***in***));

// Prompt for the 'X' player's name and initialize the player.

System.***out***.print("\nPlease enter the name of the 'X' player: ");

String name = stdin.readLine();

**while** (name == **null**) {

System.***out***.print("Please try again: ");

name = stdin.readLine();

}

xPlayer = **new** Player(name, ***LETTER\_X***);

xPlayer.setBoard(theGame.theBoard);

// Prompt for the 'O' player's name and initialize the player.

System.***out***.print("\nPlease enter the name of the 'O' player: ");

name = stdin.readLine();

**while** (name == **null**) {

System.***out***.print("Please try again: ");

name = stdin.readLine();

}

oPlayer = **new** Player(name, ***LETTER\_O***);

oPlayer.setBoard(theGame.theBoard);

// Create a referee, set the board, 'X' player, and 'O' player, then appoint the referee.

theRef = **new** Referee();

theRef.setBoard(theGame.theBoard);

theRef.setoPlayer(oPlayer);

theRef.setxPlayer(xPlayer);

theGame.appointReferee(theRef);

}

}

**CONSTANTS.JAVA**

/\*\*

\* The Constants interface defines character constants.

\*/

**public** **interface** Constants {

/\*\*

\* Represents a space character.

\*/

**static** **final** **char** ***SPACE\_CHAR*** = ' ';

/\*\*

\* Represents the letter 'O'.

\*/

**static** **final** **char** ***LETTER\_O*** = 'O';

/\*\*

\* Represents the letter 'X'.

\*/

**static** **final** **char** ***LETTER\_X*** = 'X';

}

**REFEREE.JAVA**

/\*\*

\* The Referee class manages and controls the game flow between two players on a game board.

\*/

**public** **class** Referee {

**private** Player xPlayer; // Represents the player with the 'X' symbol.

**private** Player oPlayer; // Represents the player with the 'O' symbol.

**private** Board board; // Represents the game board.

/\*\*

\* Constructs a new Referee instance.

\*/

**public** Referee() {

}

/\*\*

\* Sets the player who will use the 'X' symbol.

\*

\* **@param** xPlayer The 'X' player.

\*/

**public** **void** setxPlayer(Player xPlayer) {

**this**.xPlayer = xPlayer;

}

/\*\*

\* Sets the player who will use the 'O' symbol.

\*

\* **@param** oPlayer The 'O' player.

\*/

**public** **void** setoPlayer(Player oPlayer) {

**this**.oPlayer = oPlayer;

}

/\*\*

\* Sets the game board for the referee to manage.

\*

\* **@param** board The game board.

\*/

**public** **void** setBoard(Board board) {

**this**.board = board;

}

/\*\*

\* Runs the tic-tac-toe game.

\* The game continues until a player wins, it's a tie, or an error occurs.

\*/

**public** **void** runTheGame() {

xPlayer.setOpponent(oPlayer);

oPlayer.setOpponent(xPlayer);

System.***out***.println("\nReferee started the game...");

board.display();

Player currentPlayer = xPlayer;

**while** (**true**) {

currentPlayer.play();

board.display();

**if** (board.xWins()) {

System.***out***.println("THE GAME IS OVER: " + currentPlayer.getName() + " is the winner!");

System.***out***.println("Game Ended ...");

**break**;

} **else** **if** (board.oWins()) {

System.***out***.println("THE GAME IS OVER: " + currentPlayer.getName() + " is the winner!");

System.***out***.println("Game Ended ...");

**break**;

} **else** **if** (board.isFull()) {

System.***out***.println("THE GAME IS OVER: It's a tie!");

System.***out***.println("Game Ended ...");

**break**;

}

// Switch players for the next turn

currentPlayer = currentPlayer.getOpponent();

}

}

}

**PLAYER.JAVA**

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

/\*\*

\* The Player class represents a player in a tic-tac-toe game.

\* Each player has a name and a mark ('X' or 'O') and can make moves on the game board.

\* This class provides methods for player actions and interactions.

\* @author Chioma Ukaegbu

\*/

public class Player {

private String name; // The name of the player.

private char mark; // The mark ('X' or 'O') of the player.

private Player opponent; // The opponent player.

private Board board; // The game board.

/\*\*

\* Constructs a new Player instance with a name and a mark.

\* @param name The name of the player.

\* @param mark The player's mark ('X' or 'O').

\*/

public Player(String name, char mark) {

this.name = name;

this.mark = mark;

}

/\*\*

\* Sets the opponent player for this player.

\* @param opponent The opponent player.

\*/

public void setOpponent(Player opponent) {

this.opponent = opponent;

}

/\*\*

\* Sets the game board for this player to interact with.

\* @param board The game board.

\*/

public void setBoard(Board board) {

this.board = board;

}

/\*\*

\* Gets the opponent player.

\* @return The opponent player.

\*/

public Player getOpponent() {

return opponent;

}

/\*\*

\* Gets the name of the player.

\* @return The player's name.

\*/

public String getName() {

return name;

}

/\*\*

\* The main play method for a player.

\* Calls the makeMove() method to make a move on the game board.

\*/

public void play() {

makeMove();

}

/\*\*

\* Makes a move on the game board by taking user input for row and column.

\* Validates the move and adds the player's mark to the board.

\*/

public void makeMove() {

BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));

int row, col;

try {

do {

System.out.print(name + ", what row should your next " + mark + " be placed in? ");

row = Integer.parseInt(reader.readLine());

System.out.print(name + ", what column should your next " + mark + " be placed in? ");

col = Integer.parseInt(reader.readLine());

if (isValidMove(row, col)) {

board.addMark(row, col, mark);

break;

} else {

System.out.println("Invalid move. Try again.");

}

} while (true);

} catch (IOException e) {

e.printStackTrace();

} catch (NumberFormatException e) {

e.printStackTrace();

}

}

/\*\*

\* Checks if a move is valid by verifying that the selected row and column are within bounds

\* and the corresponding cell on the board is empty (space character).

\* @param row The selected row.

\* @param col The selected column.

\* @return true if the move is valid, false otherwise.

\*/

private boolean isValidMove(int row, int col) {

return row >= 0 && row < 3 && col >= 0 && col < 3 && board.getMark(row, col) == Constants.SPACE\_CHAR;

}

}

SAMPLE OUTPUTA screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer game

Description automatically generated