```java
import java.io.*;

/**
 * This is the Gameboard class.  It implements a two dimension array that
 * represents a connect four gameboard. It keeps track of the player making
 * the next play based on the number of pieces on the game board. It provides
 * all of the methods needed to implement the playing of a max connect four
 * game.
 *
 * @author James Spargo
 *
 */

public class GameBoard
{
    // class fields
    private int[][] playBoard;
    private int pieceCount;
    private int currentTurn;

    /**
     * This constructor creates a GameBoard object based on the input file
     * given as an argument. It reads data from the input file and provides
     * lines that, when uncommented, will display exactly what has been read
     * in from the input file.  You can find these lines by looking for
     *
     * @param inputFile the path of the input file for the game
     */
    public GameBoard( String inputFile )
    {
        this.playBoard = new int[6][7];
        this.pieceCount = 0;
        int counter = 0;
        BufferedReader input = null;
        String gameData = null;

        // open the input file
        try
        {
            input = new BufferedReader( new FileReader( inputFile ) );
        }
        catch( IOException e )
        {
            System.out.println("\nProblem opening the input file!\nTry again." +
                               "\n");
            e.printStackTrace();
        }

        //read the game data from the input file
        for(int i = 0; i < 6; i++)
        {
            try
            {
                gameData = input.readLine();

                // testing
                // uncomment the next 2 lines to see the whole line read in
                //System.out.println("I just read ->" + gameData + "<- " +
                //                   "outer for loop");

                // read each piece from the input file
                for( int j = 0; j < 7; j++ )
                {
                    //testing- uncomment the next 3 lines to see each piece
                    // that was read in
                    //System.out.println("I just read ->" +
                    //          ( gameData.charAt( counter ) - 48 ) +
                    //          "<- inner for loop");

                    this.playBoard[ i ][ j ] = gameData.charAt( counter++ ) - 48;

                    // sanity check
                    if( !( ( this.playBoard[ i ][ j ] == 0 ) ||
                           ( this.playBoard[ i ][ j ] == 1 ) ||
                           ( this.playBoard[ i ][ j ] == 2 ) ) )
                    {
                        System.out.println("\nProblems!\n--The piece read " +
                                           "from the input file was not a 1, a 2 or a 0"
);
                        this.exit_function( 0 );
                    }

                    if( this.playBoard[ i ][ j ] > 0 )
                    {
                        this.pieceCount++;
                    }
                }
            }
            catch( Exception e )
            {
                System.out.println("\nProblem reading the input file!\n" +
                                   "Try again.\n");
                e.printStackTrace();
                this.exit_function( 0 );
            }

            //reset the counter
            counter = 0;

        } // end for loop

        // read one more line to get the next players turn
        try
        {
            gameData = input.readLine();
        }
        catch( Exception e )
        {
            System.out.println("\nProblem reading the next turn!\n" +
                               "--Try again.\n");
            e.printStackTrace();
        }

        this.currentTurn = gameData.charAt( 0 ) - 48;

        //testing-uncomment the next 2 lines to see which current turn was read
        //System.out.println("the current turn i read was->" +
        //              this.currentTurn );

        // make sure the turn corresponds to the number of pcs played already
        if(!( ( this.currentTurn == 1) || ( this.currentTurn == 2 ) ) )
        {
            System.out.println("Problems!\n the current turn read is not a " +
                               "1 or a 2!");
            this.exit_function( 0 );
        }
        else if ( this.getCurrentTurn() != this.currentTurn )
        {
            System.out.println("Problems!\n the current turn read does not " +
                           "correspond to the number of pieces played!");
            this.exit_function( 0 );
        }
    } // end GameBoard( String )


    /**
     * This constructor creates a GameBoard object from another double
     * indexed array.
     *
     * @param masterGame a dual indexed array
     */
    public GameBoard( int masterGame[][] )
    {

        this.playBoard = new int[6][7];
        this.pieceCount = 0;

        for( int i = 0; i < 6; i++ )
        {
            for( int j = 0; j < 7; j++)
            {
                this.playBoard[ i ][ j ] = masterGame[ i ][ j ];

                if( this.playBoard[i][j] > 0 )
                {
                    this.pieceCount++;
                }
            }
        }
    } // end GameBoard( int[][] )

    /**
     * this method returns the score for the player given as an argument.
     * it checks horizontally, vertically, and each direction diagonally.
     * currently, it uses for loops, but i'm sure that it can be made
     * more efficient.
     *
     * @param player the player whose score is being requested.  valid
     * values are 1 or 2
     * @return the integer of the players score
     */
    public int getScore( int player )
    {
        //reset the scores
        int playerScore = 0;

        //check horizontally
        for( int i = 0; i < 6; i++ )
        {
            for( int j = 0; j < 4; j++ )
            {
                if( ( this.playBoard[ i ][j] == player ) &&
                    ( this.playBoard[ i ][ j+1 ] == player ) &&
                    ( this.playBoard[ i ][ j+2 ] == player ) &&
                    ( this.playBoard[ i ][ j+3 ] == player ) )
                {
                    playerScore++;
                }
            }
        } // end horizontal

        //check vertically
        for( int i = 0; i < 3; i++ ) {
            for( int j = 0; j < 7; j++ ) {
```

```
            if( ( this.playBoard[ i ][ j ] == player ) &&
                ( this.playBoard[ i+1 ][ j ] == player ) &&
                ( this.playBoard[ i+2 ][ j ] == player ) &&
                ( this.playBoard[ i+3 ][ j ] == player ) ) {
                playerScore++;
            }
        }
    } // end verticle

    //check diagonally - backs lash ->      \
        for( int i = 0; i < 3; i++ ){
            for( int j = 0; j < 4; j++ ) {
                if( ( this.playBoard[ i ][ j ] == player ) &&
                    ( this.playBoard[ i+1 ][ j+1 ] == player ) &&
                    ( this.playBoard[ i+2 ][ j+2 ] == player ) &&
                    ( this.playBoard[ i+3 ][ j+3 ] == player ) ) {
                    playerScore++;
                }
            }
        }

        //check diagonally - forward slash -> /
        for( int i = 0; i < 3; i++ ){
            for( int j = 0; j < 4; j++ ) {
                if( ( this.playBoard[ i+3 ][ j ] == player ) &&
                    ( this.playBoard[ i+2 ][ j+1 ] == player ) &&
                    ( this.playBoard[ i+1 ][ j+2 ] == player ) &&
                    ( this.playBoard[ i ][ j+3 ] == player ) ) {
                    playerScore++;
                }
            }
        }// end player score check

        return playerScore;
} // end getScore

/**
 * the method gets the current turn
 * @return an int value representing whose turn it is.  either a 1 or a 2
 */
public int getCurrentTurn()
{
    return ( this.pieceCount % 2 ) + 1 ;
} // end getCurrentTurn


/**
 * this method returns the number of pieces that have been played on the
 * board
 *
 * @return an int representing the number of pieces that have been played
 * on board alread
 */
public int getPieceCount()
{
    return this.pieceCount;
}

/**
 * this method returns the whole gameboard as a dual indexed array
 * @return a dual indexed array representing the gameboard
 */
public int[][] getGameBoard()
{
    return this.playBoard;
}

/**
 * a method that determines if a play is valid or not. It checks to see if
 * the column is within bounds.  If the column is within bounds, and the
 * column is not full, then the play is valid.
 * @param column an int representing the column to be played in.
 * @return true if the play is valid<br>
 * false if it is either out of bounds or the column is full
 */
public boolean isValidPlay( int column ) {

    if ( !( column >= 0 && column <= 7 ) ) {
        // check the column bounds
        return false;
    } else if( this.playBoard[0][ column ] > 0 ) {
        // check if column is full
        return false;
    } else {
        // column is NOT full and the column is within bounds
        return true;
    }
}

/**
 * This method plays a piece on the game board.
 * @param column the column where the piece is to be played.
 * @return true if the piece was successfully played<br>
 * false otherwise
 */
public boolean playPiece( int column ) {

    // check if the column choice is a valid play
    if( !this.isValidPlay( column ) ) {
        return false;
    } else {

        //starting at the bottom of the board,
        //place the piece into the first empty spot
        for( int i = 5; i >= 0; i-- ) {
            if( this.playBoard[i][column] == 0 ) {
                if( this.pieceCount % 2 == 0 ){
                    this.playBoard[i][column] = 1;
                    this.pieceCount++;

                } else {
                    this.playBoard[i][column] = 2;
                    this.pieceCount++;
                }

                //testing
                //warning: uncommenting the next 3 lines will
                //potentially produce LOTS of output
                //System.out.println("i just played piece in column ->" +
                //          column + "<-");
                //this.printGameBoard();
                //end testing

                return true;
            }
        }
        //the pgm shouldn't get here
        System.out.println("Something went wrong with playPiece()");

        return false;
    }
} //end playPiece

/*************************  solution methods ***********************/

/**
 * this method removes the top piece from the game board
 * @param column the column to remove a piece from
 */
public void removePiece( int column ) {

    // starting looking at the top of the game board,
    // and remove the top piece
    for( int i = 0; i < 6; i++ ) {
        if( this.playBoard[ i ][ column ] > 0 ) {
            this.playBoard[ i ][ column ] = 0;
            this.pieceCount--;

            break;
        }
    }

    //testing
    //WARNING: uncommenting the next 3 lines will potentially
    //produce LOTS of output
    //System.out.println("gameBoard.removePiece(). I am removing the " +
    //              "piece in column ->" + column + "<-");
    //this.printGameBoard();
    //end testing

} // end remove piece

/************************  end solution methods ************************/

/**
 * this method prints the GameBoard to the screen in a nice, pretty,
 * readable format
 */
public void printGameBoard()
{
    System.out.println(" -----------------");

    for( int i = 0; i < 6; i++ )
    {
        System.out.print(" | ");
        for( int j = 0; j < 7; j++ )
        {
            System.out.print( this.playBoard[i][j] + " " );
        }

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

    System.out.println(" -----------------");
} // end printGameBoard

/**
 * this method prints the GameBoard to an output file to be used for
 * inspection or by another running of the application
 * @param outputFile the path and file name of the file to be written
 */
public void printGameBoardToFile( String outputFile ) {
    try {
        BufferedWriter output = new BufferedWriter(
                                                   new FileWriter( outputFile ) );
        for( int i = 0; i < 6; i++ ) {
            for( int j = 0; j < 7; j++ ) {
                output.write( this.playBoard[i][j] + 48 );
            }
            output.write("\r\n");
        }

        //write the current turn
        output.write( this.getCurrentTurn() + "\r\n");
        output.close();

    } catch( IOException e ) {
        System.out.println("\nProblem writing to the output file!\n" +
                           "Try again.");
        e.printStackTrace();
    }
} // end printGameBoardToFile()

private void exit_function( int value ){
    System.out.println("exiting from GameBoard.java!\n\n");
    System.exit( value );
}

}  // end GameBoard class
```