public class ComputerConnect4Player extends Player {
    private int depth;    // Look-ahead depth

    public ComputerConnect4Player(String name, int maxDepth) {
        super(name);
        depth = maxDepth;
    }

    @Override
    public int getMove(Connect4State state, Connect4View view) {
        int move = pickMove(state, depth, -Integer.MAX_VALUE, Integer.MAX_VALUE).move;
        view.reportMove(move, state.getPlayerToMove().getName());
        return move;
    }

    private Connect4Move pickMove(Connect4State state, int depth, int low, int high) {
        Connect4Move currentMove;      // Hold current move and its value
        Connect4Move bestMove;         // Hold best move found and its value
        char[][] board = state.getBoard();
        int playerToMove = state.getPlayerNum();
        // A dummy move that will be replaced when a real move is evaluated,
        // so the column number is irrelevant.
        bestMove = new Connect4Move(Integer.MIN_VALUE, 0);
        // Run through possible moves
        for (int c= 1; c <= Connect4Game.COLS; c++) {
            if (state.isValidMove(c)) {   // See if legal move
                Connect4Game copy = new Connect4Game(playerToMove, state.getPlayers(), board);
                copy.makeMove(c);             // Make the move
                if (copy.gameIsOver()) {
                    currentMove = new Connect4Move(Integer.MAX_VALUE, c);
                } else if (playerToMove == copy.getPlayerNum()) {    // Did current player change?
                    currentMove = pickMove(copy, depth, low, high);  // No, so no depth change
                    currentMove.move = c;              // Remember move made
                } else if (depth > 0) {          // Player changed, so reduce search depth
                    currentMove = pickMove(copy, depth - 1, -high, -low);
                    currentMove.value = -currentMove.value;   // Good for opponent is bad for me
                    currentMove.move = c;                   // Remember move made
                } else // Depth exhausted, so estimate who is winning by comparing kalahs
                    currentMove = new Connect4Move(copy.score(), c);
                if (currentMove.value > bestMove.value)  {  // Found a new best move?
                    bestMove = currentMove;
                    low = Math.max(low, bestMove.value);   // Update the low value, also
                }
            }
        }
        return bestMove;
    }
}

public class ComputerConnect4 extends Game {
    public static void main(String... args) {
        Connect4View view = new Connect4ViewText();
        Connect4View view = new Connect4ViewGraphical();
        Player[] players = new Player[2];                // Array to hold the players
        // Initialize the players
        players[0] = makePlayer(view, "first");
    }
}
players[1] = makePlayer(view, "second");
  // Hold current game state
  Connect4Game state = new Connect4Game(playerNum, players);
  view.display(state);
  while (!state.gameIsOver()) {
    int move = state.getPlayerToMove().getMove(state, view);
    state.makeMove(move);
    view.display(state);
  }
  if (state.isFull())
    view.reportToUser("It is a draw");
  else
    view.reportToUser(players[1 - state.getPlayerNum()].getName() + " wins!");
  }

  public static Player makePlayer(Connect4View view, String playerMsg) {
    String playerName = view.getAnswer("Enter the name of the " + playerMsg + " player.
(Include 'Computer' in the name of a computer player) ");
    if(playerName.contains("Computer")) {
      int depth = view.getIntAnswer("How far should I look ahead? ");
      return new ComputerConnect4Player(playerName, depth);
    }
    else
      return new HumanConnect4Player(playerName);
  }

  public class Connect4Game implements Connect4State {
    private char[][] board;
    private int playerToMoveNum;    // 0 or 1 for first and second player
    private Player[] players;      // Array of the two players
    //private Connect4View view;
    public final static int ROWS = 6;            // Board height
    public final static int COLS = 7;            // Board width
    public final static char EMPTY = '.';        // Indicate empty place
    public final static char CHECKER0 = 'X';     // Indicate the first player's checker
    public final static char CHECKER1 = 'O';     // Indicate second player's checker
    public final static char[] CHECKERS = {CHECKER0, CHECKER1};

    public Connect4Game(int playerNum, Player[] thePlayers) {
      char[][] initBoard = new char[ROWS][COLS];
      for (int j = 0; j < ROWS; j++) {
        for (int k = 0; k < COLS; k++) {
          initBoard[j][k] = '.';
        }
      }
      board = initBoard;
      playerToMoveNum = playerNum;
      players = thePlayers;
    //view = aView;  // needed?
    }

    public Connect4Game(int playerNum, Player[] thePlayers, char[][] initBoard) {
      board = new char[ROWS][COLS];
      for (int i = 0; i < ROWS; i++) {
        for (int j = 0; j < COLS; j++) {
          board[i][j] = initBoard[i][j];
        }
      }
      playerToMoveNum = playerNum;
      players = thePlayers;
    }

    public char[][] getBoard() {
      return board;
    }
  }
/**
 * Gets an array holding 2 Player objects
 * @return the players
 */
public Player[] getPlayers() {
    return players;
}

/**
 * Gets the number of the player whose move it is
 * @return the number of the player whose move it is
 */
public int getPlayerNum() {
    return playerToMoveNum;
}

/**
 * Gets the Player whose turn it is to move
 * @return the Player whose turn it is to move
 */
public Player getPlayerToMove() {
    return players[playerToMoveNum];
}

/**
 * Is this move valid?
 * @param col column where we want to move
 * @return true if the move is valid
 */
public boolean isValidMove(int col) {
    if (board[ROWS-1][col-1] == EMPTY) {
        return true;
    } else {
        return false;
    }
}

/**
 * Make a move, dropping a checker in the given column
 * @param col the column to get the new checker
 */
public void makeMove(int col) {
    int r = 0;
    while (board[r][col-1] != EMPTY && r < ROWS) {
        r++;
    }
    if (r >= ROWS)
    board[r][col-1] = CHECKERS[playerToMoveNum];
    playerToMoveNum = 1 - playerToMoveNum;
}

/**
 * Is the board full?
 * @return true if the board is full
 */
public boolean isFull() {
    boolean full = true;
    for (int c = 0; c < COLS; c++) {
        if (board[ROWS-1][c] == EMPTY) {
            full = false;
        }
    }
    return full;
}

/**
 * Decide if the game is over
 * @return true if the game is over
 */
public boolean gameIsOver() {
    boolean gameOver = false;
    if (isFull()) {
        gameOver = true;
    } else if (connectFourAnywhere()) {
        gameOver = true;
    }
    return gameOver;
}

/**
 * Get the score of a board
 */
public int score() {
    int score = 0;
    for (int r = 0; r < ROWS; r++) {
        if (r <= ROWS-4) {
            for (int c = 0; c < COLS; c++) {
                score += score(r, c);
            }
        } else {
            for (int c = 0; c <= COLS-4; c++) {
                score += score(r, c);
            }
        }
    }
    return score;
}

/**
 * Helper method to get the score of a board
 */
public int score(int row, int col) {
    int score = 0;
    boolean unblocked = true;
    int tally = 0;
    //check up
    if (row < ROWS-3) {
        if (checkUpUnblocked) {
            return tally;
        } else {
            tally += score(row + 1, col);
        }
    } else {
        if (checkUpUnblocked) {
            return tally;
        } else {
            tally += score(row - 1, col);
        }
    }
    return tally;
}
for (int r=row; r<row+4; r++) {
    if (board[r][col] == CHECKERS[1-playerToMoveNum]) {
        unblocked = false;
    }
    if (board[r][col] == CHECKERS[playerToMoveNum]) {
        tally ++;
    }
    if (unblocked == true) {
        score = score + (tally*tally*tally*tally);
    }
}
if (col < COLS-3) {
    // check up and to the right
    unblocked = true;
    tally = 0;
    for (int r=row, c=col; r<row+4; r++, c++) {
        if (board[r][c] == CHECKERS[1-playerToMoveNum]) {
            unblocked = false;
        }
        if (board[r][c] == CHECKERS[playerToMoveNum]) {
            tally ++;
        }
        if (unblocked == true) {
            score = score + (tally*tally*tally*tally);
        }
    }
}
if (col < COLS-3) {
    // check right
    unblocked = true;
    tally = 0;
    for (int c=col; c<col+4; c++) {
        if (board[row][c] == CHECKERS[1-playerToMoveNum]) {
            unblocked = false;
        }
        if (board[row][c] == CHECKERS[playerToMoveNum]) {
            tally ++;
        }
        if (unblocked == true) {
            score = score + (tally*tally*tally*tally);
        }
    }
}
if (row > 2) {
    // check down and to the right
    unblocked = true;
    tally = 0;
    for (int r=row, c=col; c<col+4; r--, c++) {
        if (board[r][c] == CHECKERS[1-playerToMoveNum]) {
            unblocked = false;
        }
        if (board[r][c] == CHECKERS[playerToMoveNum]) {
            tally ++;
        }
        if (unblocked == true) {
            score = score + (tally*tally*tally*tally);
        }
    }
}
return score;

/**
 * Check the board to see if there is a Connect Four anywhere
 * @return true if there is a connect 4 somewhere
 */
public boolean connectFourAnywhere() {
    boolean connect4 = false;
    for (int r=0; r < ROWS && connect4 == false; r++) {
        if (r <= ROWS-4) {
            for (int c = 0; c < COLS && connect4 == false; c++) {
                connect4 = connectFour(r, c);
            }
        } else {
            for (int c = 0; c <= COLS-4 && connect4 == false; c++) {
                connect4 = connectFour(r, c);
            }
        }
    }
    return connect4;
}

/**
 * Given a row and column, check for a connect 4 from that position
 * @return true if there is a connect 4.
 */
public boolean connectFour(int row, int col) {
    boolean c4 = false;
    // check up
    c4 = true;
    for (int r=row; r<row+4; r++) {
        if (board[r][col] != CHECKERS[1-playerToMoveNum]) {
            c4 = false;
        }
    }
    if (col < COLS-3 && c4==false) {
        // check up and to the right
        c4 = true;
        for (int r=row, c=col; r<row+4; r++, c++) {
            if (board[r][c] != CHECKERS[1-playerToMoveNum]) {
                c4 = false;
            }
        }
    }
if (col < COLS-3 && c4==false) {
    //check right
    c4 = true;
    for (int c=col; c<col+4; c++) {
        if (board[row][c] != CHECKERS[1-playerToMoveNum]) {
            c4 = false;
        }
    }
}
if (row > 2 && c4==false) {
    //check down and to the right
    c4 = true;
    for (int r=row, c=col; c<col+4; r--, c++) {
        if (board[r][c] != CHECKERS[1-playerToMoveNum]) {
            c4 = false;
        }
    }
}
return c4;

/**
 * A class which a move and its value to be returned from a call to a method.
 * @author Ryan Maguire
 */
public class Connect4Move {
    public int value;       // Game value of this move
    public int move;        // Number of pit to be emptied
    public Connect4Move(int value, int move) {
        this.value = value;
        this.move = move;
    }
}

/**
 * An interface for the game state of the Connect 4 game.
 * @author Scot Drysdale
 */
public interface Connect4State {
    final static int ROWS = 6;            // Board height
    final static int COLS = 7;            // Board width
    final static char EMPTY = '.';        // Indicate empty place
    final static char CHECKER0 = 'X';     // Indicate the first player's checker
    final static char CHECKER1 = 'O';     // Indicate second player's checker
    final static char[] CHECKERS = {CHECKER0, CHECKER1};
    //
    // Gets a 2-D array representing the board.
    // The first subscript is the row number and the second the column number.
    // The bottom of the board is row 0 and the top is row ROWS-1.
    // The left side of the board is column 0 and the right side is column COLS-1.
    // Return the board
    char[][] getBoard();
    //
    // Gets an array holding 2 Player objects
    // Return the players
    Player[] getPlayers();
    //
    // Gets the number of the player whose move is it
    // Return the number of the player whose move it is
    int getPlayerNum();
    //
    // Gets the Player whose turn it is to move
    // Return the Player whose turn it is to move
    Player getPlayerToMove();
    //
    // Is this move valid?
    // Move in column where we want to move
    // Return true if the move is valid
    //
    boolean isValidMove(int col);
    //
    // Make a move, dropping a checker in the given column
    // Move in column to get the new checker
    //
    void makeMove(int col);
    //
    // Is the board full?
    // Return true if the board is full
    //
    boolean isFull();
    //
    // Decide if the game is over
    // Return true if the game is over
    //
    boolean gameIsOver();
}
public interface Connect4View {

    // Displays the current board
    // Use state current state of the game
    public void display (Connect4State state);

    // Asks the user for a move
    // The move will be in the range 0 to Connect4State.COLS-1.
    // Use state current state of the game
    // Return the number of the move that player chose
    public int getUserMove(Connect4State state);

    // Reports the move that a player has made.
    // The move should be in the range 0 to Connect4State.COLS-1.
    // Use state current state of the game
    // Return the message to be reported
    public void reportMove (int chosenMove, String name);

    // Ask the user the question and return the answer as an int
    // Use state current state of the game
    // Return the message to be reported
    public int getIntAnswer (String question);

    // Display a message to user
    // Use state current state of the game
    public void reportToUser(String message);

    // Ask the question and return the answer
    // Use state current state of the game
    // Return the answer to the question
    public String getAnswer(String question);
}

public class Connect4ViewText implements Connect4View {
    private Scanner input; // Hold the input stream

    public Connect4ViewText () {
        input = new Scanner(System.in);
    }

    // Displays the current board
    // Use state current state of the game
    public void display (Connect4State state) {
        char[][] board = state.getBoard();
        System.out.println();
        for (int r = Connect4Game.ROWS-1; r>=0; r--) {
            System.out.println(board[r][0] + " " + board[r][1] + " " + board[r][2] + " " + board[r][3] + " " + board[r][4] + " " + board[r][5] + " " + board[r][6]);
        }
    }

    // Asks the user for a move
    // The move will be in the range 0 to Connect4State.COLS-1.
    // Use state current state of the game
    // Return the number of the move that player chose
    public int getUserMove(Connect4State state) {
        int col; // The pit under consideration
        char[][] board = state.getBoard();
        System.out.println();
        System.out.print("Choose a column to drop a checker into, ");
        col = getIntAnswer("Choose a column to drop a checker into, ");
        if (col < 1 || col > Connect4Game.COLS) {
            System.out.println("Invalid column. Try again.");
            col = getIntAnswer("Choose a column to drop a checker into, ");
        }
        return col; // Adjust to player's side
    }
}
public void reportMove(int chosenMove, String name) {
    System.out.println(name + " drops a Checker in Column " + chosenMove);
}

public int getIntAnswer(String question) {
    int answer = 0;
    boolean valid = false;
    // Ask for a number
    System.out.print(question + " ");
    while(!valid) {
        try {
            answer = input.nextInt();
            valid = true; // If got to here we have a valid integer
        } catch(InputMismatchException ex) {
            reportToUser("That was not a valid integer");
            valid = false;
            input.nextLine(); // Throw away the rest of the line
        }
    }
    input.nextLine(); // Throw away the rest of the line
    return answer;
}

public void reportToUser(String message) {
    // Reports something to the user
    System.out.println(message);
}

public String getAnswer(String question) {
    System.out.print(question);
    return input.nextLine();
}

public class HumanConnect4Player extends Player {

    public HumanConnect4Player(String name) {
        super(name);
    }

    @Override
    public int getMove(Connect4State state, Connect4View view) {
        // Get a move for the user
        return view.getUserMove(state);
    }
}

public abstract class Player {
    private String playerName;

    public Player(String name) {
        playerName = name;
    }

    public String getName() {
        return playerName;
    }

    public abstract int getMove(Connect4State state, Connect4View view);
}