Super Tic-Tac-Toe Project

Russ Johnson
Grand Valley State University
Allendale, MI
russjohnson09@gmail.com

October 10, 2012

Listing 1: SuperTicTacToe.java

```
/*****************************************************************
 * Contains the main method that runs SuperTicTacToePanel.
 *
 * @author Russ Johnson
 * @version 10.10.2012
 *****************************************************************/

package package1;

import java.awt.Dimension;

import javax.swing.JFrame;
import javax.swing.JOptionPane;

public class SuperTicTacToe {

    /*****************************************************************
     * The main method starts up the GUI application.
     *
     * @param args
     *            arguments that main takes
     *
     * @return none
     *****************************************************************/
    public static void main(String[] args) {

        int size;

        // Asks user for size of board. 3 <= board <= 9.
        String sizeinput = JOptionPane.showInputDialog(null,
                "Enter in the size of the board: (Between 3 and 9 inclusive.");

        // Tries to parse integer provided by user. If not an integer or not in
        // the range 3 to 9.
        try {
            size = Integer.parseInt(sizeinput);
            if (!(2 < size && size < 10)) {
                throw new Throwable();
            }
        } catch (Throwable e) {
            JOptionPane.showMessageDialog(null,
                    "Invalid input. Default 3 will be used.");
            size = 3;
        }

        // Asks who should move first.
        String player = JOptionPane.showInputDialog("Who moves first? X or O");

        // Holds 0 if O moves first and 1 if X moves first.
        int playerint;

        // Tries to check input for X or O. If cancel is pressed or invalid
        // input, defaults to X.
        try {
            if (player.equalsIgnoreCase("X")) {
                playerint = 1;
            } else if (player.equalsIgnoreCase("O")) {
                playerint = 0;
            } else {
                JOptionPane.showMessageDialog(null,
                        "Invalid input. Default X will be used.");
                playerint = 1;
            }
        } catch (Throwable e) {
            JOptionPane.showMessageDialog(null,
                    "Invalid input. Default X will be used.");
            playerint = 1;
        }

        JFrame frame = new JFrame("Super TicTacToe");

        frame.setPreferredSize(new Dimension(600, 600));

        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        frame.getContentPane().add(new SuperTicTacToePanel(size, playerint));
        frame.pack();
        frame.setVisible(true);
        frame.setResizable(false);

    }

}
```

Listing 2: SuperTicTacToeGame.java

```
/****************************************************************
 * Controls the games logic. Implements java.io.Serializable so that it can
 * easily be saved.
 *
 * @author Russ Johnson
 * @version 10.10.2012
 ****************************************************************/

package package1;

import java.awt.Point;
import java.util.Stack;

public class SuperTicTacToeGame implements java.io.Serializable {

    /** A 2D array of Cells. Represents game's board. */
    private Cell[][] board;

    /** static variable that represents total wins for X */
    private static int xwins = 0;

    /** static variable that represents total wins for O */
    private static int owins = 0;

    /** stores the total wins for X when an instance of the class is saved */
    private int xwin;

    /** stores the total wins for O when an instance of the class is saved */
    private int owin;

    /** player's move (0 for O and 1 for X) */
    private int player;

    /** size of board */
    private int size;

    /** keeps track of each move made */
    private Stack<Point> moves = new Stack<Point>();

    /****************************************************************
     * Constructor for SuperTicTacToeGame
     ****************************************************************/
    public SuperTicTacToeGame(int player, int size) {
```

```
    board = new Cell[size][size];
    this.player = player;
    this.size = size;
    xwin = xwins;
    owin = owins;
    reset();

}

/*****************************************************************
 * Get method for player.
 *
 * @return player player for game
 *****************************************************************/
public int getPlayer() {
    return player;
}

/*****************************************************************
 * Get method for size.
 *
 * @return size the size of board
 *****************************************************************/
public int getSize() {
    return size;
}

/*****************************************************************
 * Get method for moves.
 *
 * @return moves this contains all moves taken
 *****************************************************************/
public Stack<Point> getMoves() {
    return moves;
}

/*****************************************************************
 * Get method for xwin.
 *
 * @return xwin this mirrors the static variable xwins
 *****************************************************************/
public int getXwin() {
    return xwin;
}
```

```
/*****************************************************************
 * Get method for owin.
 *
 * @return owin this mirrors the static varibale owins
 *****************************************************************/
public int getOwin() {
    return owin;
}

/*****************************************************************
 * Get method for xwins.
 *
 * @return xwins total wins for X
 *****************************************************************/
public static int getXwins() {
    return xwins;
}

/*****************************************************************
 * Get method for owins.
 *
 * @return owins total wins for O
 *****************************************************************/
public static int getOwins() {
    return owins;
}

/*****************************************************************
 * Set method for xwins.
 *
 * @param xwins
 *            set total wins for X (xwins)
 *
 * @return none
 *****************************************************************/
public static void setXwins(int xwins) {
    SuperTicTacToeGame.xwins = xwins;
}

/*****************************************************************
 * Set method for owins.
 *
 * @param set
 *            total wins for O (owins)
 *
 * @return none
 *****************************************************************/
public static void setOwins(int owins) {
    SuperTicTacToeGame.owins = owins;
}

/*****************************************************************
 * Selects a space on the board to move and sets it if valid.
 *
 * @param row
 *            select row of board
 *
 * @param col
 *            select column of board
 *
 * @return none
 *****************************************************************/
public void select(int row, int col) {
    if (isvalidmove(row, col)) {
        moves.push(new Point(row, col));

        if (player == 0) {
            board[row][col] = Cell.O;
        } else {
            board[row][col] = Cell.X;
        }
        nextPlayer();
    }

}

/*****************************************************************
 * Selects a space on the board to move and sets it if valid.
 *
 * @param p
 *            selects point of board
 *
 * @return none
 *****************************************************************/
public void select(Point p) {
    int row = p.x;
    int col = p.y;
    if (isvalidmove(row, col)) {
        moves.push(p);
        if (player == 0) {
            board[row][col] = Cell.O;
        } else {
            board[row][col] = Cell.X;
        }
        nextPlayer();
    }

}

/*****************************************************************
 * Returns a boolean for whether or not a move is valid.
 *
 * @param row
 *            selects row of board
 *
 * @param col
 *            selects column of board
 *
 * @return boolean returns True if valid move else otherwise
 *****************************************************************/
public boolean isvalidmove(int row, int col) {
    return board[row][col] == Cell.EMPTY;
}

/*****************************************************************
 * Undoes a move.
 *
 * @return none
 *****************************************************************/
public void undo() {
    if (undoIsValid()) {
        Point move = moves.pop();
        board[move.x][move.y] = Cell.EMPTY;
        nextPlayer();
    }
}

/*****************************************************************
 * Returns a boolean for whether or not undo is valid.
 *
 * @return boolean returns true if valid undo false otherwise
 *****************************************************************/
public boolean undoIsValid() {
    return !(moves.isEmpty());
```

```
}

/*****************************************************************
 * Resets the board.
 *
 * @return none
 *****************************************************************/
public void reset() {
    for (int row = 0; row < size; row++) {
        for (int col = 0; col < size; col++) {
            board[row][col] = Cell.EMPTY;
        }
    }

}

/*****************************************************************
 * Sets the player to the next player.
 *
 * @return none
 *****************************************************************/
private void nextPlayer() {
    player = (player + 1) % 2;
}

/*****************************************************************
 * Returns the status of the game and increment owins or xwins if either
 * wins.
 *
 * @return GameStatus Status of the game.
 *****************************************************************/
public GameStatus getGameStatus() {
    if (moves.size() > 4) {
        Cell pCell;
        int row = moves.lastElement().x;
        int col = moves.lastElement().y;
        if (player == 0) {
            pCell = Cell.X;
        } else {
            pCell = Cell.O;

        }
        if (isHorizontal(pCell, row, col) || isVertical(pCell, row, col)
                || isDiagonal(pCell, row, col)) {
            if (pCell == Cell.O) {
                owins++;
                return GameStatus.O_WON;
            } else {
                xwins++;
                return GameStatus.X_WON;
            }
        } else if (boardIsFull()) {
            return GameStatus.CATS;

        } else {
            return GameStatus.IN_PROGRESS;
        }
    }
    return GameStatus.IN_PROGRESS;
}

/*****************************************************************
 * Checks for a three in a row diagonally.
 *
 * @return boolean true if diagonal is found false otherwise
 *****************************************************************/
private boolean isDiagonal(Cell pCell, int row, int col) {
    return (isDiagonalUpperLeft(pCell, row, col) || isDiagonalUpperRight(
            pCell, row, col));
}

/*****************************************************************
 * Checks for a three in a row diagonally from the upper left.
 *
 * @return boolean true if diagonal is found false otherwise
 *****************************************************************/
private boolean isDiagonalUpperLeft(Cell pCell, int row, int col) {
    int count = 1;
    while (true) {
        row--;
        col--;
        if (row > -1 && col > -1 && board[row][col] == pCell) {
            count++;
        } else {
            break;
        }
    }
    row += count;
    col += count;
    while (true) {
        row++;
        col++;
        if (row < size && col < size && board[row][col] == pCell) {
            count++;
        } else {
            break;
        }
    }
    if (count > 2) {
        return true;
    } else {
        return false;
    }
}

/****************************************************************
 * Checks for a three in a row diagonally from the upper right.
 *
 * @return boolean true if diagonal is found false otherwise
 ****************************************************************/
private boolean isDiagonalUpperRight(Cell pCell, int row, int col) {
    int count = 1;
    while (true) {
        row++;
        col--;
        if (row < size && col > -1 && board[row][col] == pCell) {
            count++;
        } else {
            break;
        }
    }
    row -= count;
    col += count;
    while (true) {
        row--;
        col++;
        if (row > -1 && col < size && board[row][col] == pCell) {
            count++;
        } else {
            break;
        }
    }
    if (count > 2) {
        return true;
    } else {
        return false;
    }
}

/*****************************************************************
 * Checks for a three in a row vertically.
 *
 * @return boolean true if vertical is found false otherwise
 *****************************************************************/
private boolean isVertical(Cell pCell, int row, int col) {
    int count = 1;
    while (true) {
        row--;
        if (row > -1 && board[row][col] == pCell) {
            count++;
        } else {
            break;
        }
    }
    row += count;
    while (true) {
        row++;
        if (row < size && board[row][col] == pCell) {
            count++;
        } else {
            break;
        }
    }
    if (count > 2) {
        return true;
    } else {
        return false;
    }
}

/*****************************************************************
 * Checks for a three in a row horizontally.
 *
 * @return boolean true if horizontal is found false otherwise
 *****************************************************************/
private boolean isHorizontal(Cell pCell, int row, int col) {
    int count = 1;
    while (true) {
        col--;
        if (col > -1 && board[row][col] == pCell) {
            count++;
        } else {
            break;
        }
    }
    col += count;
    while (true) {
        col++;
        if (col < size && board[row][col] == pCell) {
            count++;
        } else {
            break;
        }
    }
    if (count > 2) {
        return true;
    } else {
        return false;
    }
}

/*****************************************************************
 * Checks for the board being full.
 *
 * @return boolean true if board is full false otherwise
 *****************************************************************/
private boolean boardIsFull() {
    for (Cell[] row : board) {
        for (Cell cell : row) {
            if (cell == Cell.EMPTY) {
                return false;
            }
        }
    }
    return true;
}

/*****************************************************************
 * Get method for board.
 *
 * @return board board for game
 *****************************************************************/
public Cell[][] getBoard() {
    return board;
}
```

```
/*****************************************************************
 * The computer makes a move.
 *
 * @return Point point on the board
 *****************************************************************/
public Point computersmove() {
    Point move;
    move = winningmove();
    if (move.x != -1) {
        return move;
    }
    move = blockingmove();
    if (move.x != -1) {
        return move;
    } else {
        return dumbmove();
    }

}

/*****************************************************************
 * Returns a point that is the first available move.
 *
 * @return Point point on the board
 *****************************************************************/
private Point dumbmove() {
    for (int row = 0; row < size; row++) {
        for (int col = 0; col < size; col++) {
            if (isvalidmove(row, col)) {
                select(row, col);
                return new Point(row, col);
            }
        }
    }
    return new Point(-1, -1);
}

/*****************************************************************
 * Finds a winning move.
 *
 * @return Point point on the board
 *****************************************************************/
private Point winningmove() {
    for (int row = 0; row < size; row++) {
        for (int col = 0; col < size; col++) {
            if (isvalidmove(row, col)) {
                select(row, col);
                if (getGameStatusNoInc() != GameStatus.IN_PROGRESS) {
                    undo();
                    return new Point(row, col);
                } else {
                    undo();
                }
            }
        }
    }

    return new Point(-1, -1);
}

/*******************************************************************
 * Finds a blocking move. This is a move that if not taken could cause the
 * opponent to win in their next move.
 *
 * @return Point point on the board
 *******************************************************************/
private Point blockingmove() {
    nextPlayer();
    Point move = winningmove();
    nextPlayer();
    return move;

}

/*******************************************************************
 * Returns the status of the game. Same as getGameStatus but does not
 * increment the wins. Needed for computersmove.
 *
 * @return GameStatus status of the game
 *******************************************************************/
public GameStatus getGameStatusNoInc() {
    if (moves.size() > 4) {
        Cell pCell;
        int row = moves.lastElement().x;
        int col = moves.lastElement().y;
        if (player == 0) {
            pCell = Cell.X;
        } else {
            pCell = Cell.O;
        }
        if (isHorizontal(pCell, row, col) || isVertical(pCell, row, col)
                || isDiagonal(pCell, row, col)) {
            if (pCell == Cell.O) {
                return GameStatus.O_WON;
            } else {
                return GameStatus.X_WON;
            }
        } else if (boardIsFull()) {
            return GameStatus.CATS;

        } else {
            return GameStatus.IN_PROGRESS;
        }
    }
    return GameStatus.IN_PROGRESS;
}
}
```

Listing 3: SuperTicTacToePanel.java

```
/****************************************************************
 * Contains all of the GUI components need to run the SuperTicTacToeGame.
 *
 * @author Russ Johnson
 * @version 10.10.2012
 ****************************************************************/

package package1;

import java.awt.BorderLayout;
import java.awt.FlowLayout;
import java.awt.GridLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.io.PrintWriter;

import javax.swing.ImageIcon;
import javax.swing.JButton;
import javax.swing.JComponent;
import javax.swing.JFileChooser;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JPanel;

public class SuperTicTacToePanel extends JPanel {

    /** game's board made up of a 2D array of JButtons */
    private JButton[][] board;

    /** holds the size of the board */
    private int size;

    /** JLabel for scoreboard */
    private JLabel xwins;

    /** JLabel for scoreboard */
    private JLabel owins;
```

```java
/** JButton to quit gui */
private JButton quitButton;

/** JButton to undo move or moves */
private JButton undo;

/** JButton to load game */
private JButton load;

/** JButton to save game */
private JButton save;

/** JButton for computer to make move */
private JButton ai;

/** Holds an ImageIcon for X's move that is latter resized */
private ImageIcon xIconOriginal = new ImageIcon("x.png");

/** Holds an ImageIcon for O's move that is latter resized */
private ImageIcon oIconOriginal = new ImageIcon("o.png");

/** Holds the resized ImageIcon for X that can be used in the GUI. */
private ImageIcon xIcon;

/** Holds the resized ImageIcon for O that can be used in the GUI. */
private ImageIcon oIcon;

/** Empty ImageIcon for available spaces on board. */
private ImageIcon emptyIcon = new ImageIcon();

/** Instance of SuperTicTacToeGame used for the logic of the game. */
private SuperTicTacToeGame game;

/** 2D array of Cells that represent moves taken. */
private Cell[][] iBoard;

/** JPanel that contains the JButtons quitbutton, load, save, undo, and ai. */
private JPanel top;

/** JPanel that contains the 2D Array of JButtons board. */
private JPanel bottom;

/** JPanel that contains the two JLabels xwins and owins. */
private JPanel scoreboard;
```

```java
/** Instance of inner class that listens for buttons. */
ButtonListener listener = new ButtonListener();

/*****************************************************************
 * Constructor for the SuperTicTacToePanel.
 *****************************************************************/
SuperTicTacToePanel(int size, int player) {

    this.size = size;

    // Starts up the JPanels
    top = new JPanel();
    top.setLayout(new GridLayout(1, 5));
    bottom = new JPanel();
    bottom.setLayout(new GridLayout(size, size));
    scoreboard = new JPanel();
    scoreboard.setLayout(new FlowLayout());

    // Sets scoreboard to zero.
    xwins = new JLabel("X: 0");
    owins = new JLabel("O: 0");

    // Add xwins and owins to scoreboard.
    scoreboard.add(xwins);
    scoreboard.add(owins);

    // Sets up and quitButton, undo, load, save, and ai
    // and adds them to top JPanel.
    quitButton = new JButton("Quit");
    top.add(quitButton);
    undo = new JButton("Undo");
    top.add(undo);
    load = new JButton("Load");
    top.add(load);
    save = new JButton("Save");
    top.add(save);
    ai = new JButton("AI");
    top.add(ai);

    // Adds the listeners to the JButtons.
    quitButton.addActionListener(listener);
    undo.addActionListener(listener);
    load.addActionListener(listener);
    save.addActionListener(listener);
```

```
        ai.addActionListener(listener);

        // Instantiates the board using size.
        board = new JButton[size][size];

        // Instantiates each JButton in board and adds listener.
        for (int row = 0; row < size; row++) {
            for (int col = 0; col < size; col++) {
                board[row][col] = new JButton();
                bottom.add(board[row][col]);
                board[row][col].addActionListener(listener);
            }
        }

        // Adds JPanels to SuperTicTacToePanel. this.setLayout() etc. is
        // implied.
        setLayout(new BorderLayout());
        add(BorderLayout.NORTH, top);
        add(BorderLayout.CENTER, bottom);
        add(BorderLayout.SOUTH, scoreboard);

        // Sets up board with "player" starting and having size "size".
        game = new SuperTicTacToeGame(player, size);

        // Sets up iBoard
        iBoard = game.getBoard();

        // Instantiates xIcon and oIcon of correct size.
        xIcon = new ImageIcon(xIconOriginal.getImage().getScaledInstance(
                600 / size, 600 / size, java.awt.Image.SCALE_SMOOTH));

        oIcon = new ImageIcon(oIconOriginal.getImage().getScaledInstance(
                600 / size, 600 / size, java.awt.Image.SCALE_SMOOTH));

    }

    /*****************************************************************
     * Sets up the iBoard.
     *
     * @return none
     *****************************************************************/
    private void displayBoard() {
        iBoard = game.getBoard();

        for (int row = 0; row < size; row++) {
            for (int col = 0; col < size; col++) {
                if (iBoard[row][col] == Cell.O) {
                    board[row][col].setIcon(oIcon);
                } else if (iBoard[row][col] == Cell.X) {
                    board[row][col].setIcon(xIcon);
                } else {
                    board[row][col].setIcon(emptyIcon);
                }
            }
        }
    }

    /*****************************************************************
     * Resets the iBoard and game.
     *
     * @return none
     *****************************************************************/
    private void reset() {
        String player = JOptionPane.showInputDialog("Who moves first? X or O");

        int playerint;

        try {
            if (player.equalsIgnoreCase("X")) {
                playerint = 1;
            } else if (player.equalsIgnoreCase("O")) {
                playerint = 0;
            } else {
                JOptionPane.showMessageDialog(null,
                        "Invalid input. Default X will be used.");
                playerint = 1;
            }
        } catch (Throwable e) {
            JOptionPane.showMessageDialog(null,
                    "Invalid input. Default X will be used.");
            playerint = 1;
        }
        game = new SuperTicTacToeGame(playerint, size);

        xwins.setText("X: " + SuperTicTacToeGame.getXwins());
        owins.setText("O: " + SuperTicTacToeGame.getOwins());

        displayBoard();
    }
```

```
/*****************************************************************
 * Loads a new game and sets up the GUI based on this game.
 *
 * @param game
 *            this.game is set to game and then used to reconstruct GUI
 *
 * @return none
 *****************************************************************/
private void reload(SuperTicTacToeGame game) {
    this.game = game;
    SuperTicTacToeGame.setOwins(game.getOwin());
    SuperTicTacToeGame.setXwins(game.getXwin());

    size = game.getSize();
    bottom.removeAll();

    board = new JButton[size][size];
    bottom.setLayout(new GridLayout(size, size));

    for (int row = 0; row < size; row++) {
        for (int col = 0; col < size; col++) {
            board[row][col] = new JButton("");
            bottom.add(board[row][col]);
            board[row][col].addActionListener(listener);
        }
    }

    bottom.validate();

    iBoard = game.getBoard();

    xIcon = new ImageIcon(xIconOriginal.getImage().getScaledInstance(
            600 / size, 600 / size, java.awt.Image.SCALE_SMOOTH));

    oIcon = new ImageIcon(oIconOriginal.getImage().getScaledInstance(
            600 / size, 600 / size, java.awt.Image.SCALE_SMOOTH));

    xwins.setText("X: " + SuperTicTacToeGame.getXwins());
    owins.setText("O: " + SuperTicTacToeGame.getOwins());

}

/*****************************************************************
 * Inner listener class.
 *****************************************************************/
```

```
private class ButtonListener implements ActionListener {

    /*****************************************************************
     * Takes appropriate action based on the JButton pressed.
     *
     * @param event
     *            holds the event that took place
     *
     * @return none
     *****************************************************************/
    public void actionPerformed(ActionEvent event) {

        JComponent comp = (JComponent) event.getSource();

        /** Checks all of the JBottons in board. */
        for (int row = 0; row < size; row++) {
            for (int col = 0; col < size; col++) {
                if (board[row][col] == comp) {
                    game.select(row, col);
                }
            }

        }

        if (comp == undo) {
            game.undo();
        }

        if (comp == ai) {
            game.select(game.computersmove());
        }

        if (comp == load) {
            try {

                JFileChooser chooser = new JFileChooser();

                int status = chooser.showOpenDialog(null);

                if (status != JFileChooser.APPROVE_OPTION)
                    System.out.println("No File Chosen");
                else {
                    File file = chooser.getSelectedFile();
                    FileInputStream fileIn = new FileInputStream(file);
                    ObjectInputStream in = new ObjectInputStream(fileIn);
                    SuperTicTacToeGame game1 = (SuperTicTacToeGame) in
                            .readObject();
                    in.close();
                    fileIn.close();
                    reload(game1);
                }

            } catch (IOException i) {
                i.printStackTrace();
            } catch (ClassNotFoundException c) {

            }
        }

        // Board is displayed after all of board's buttons and the undo, ai,
        // and load buttons have been checked.
        displayBoard();

        if (comp == quitButton
                && JOptionPane.showConfirmDialog(null, "Are you sure?") == 0) {
            System.exit(1);

        }

        if (comp == save) {
            try {
                PrintWriter out = null;
                JFileChooser chooser = new JFileChooser();
                int status = chooser.showOpenDialog(null);

                if (status != JFileChooser.APPROVE_OPTION)
                    System.out.println("No File Chosen");
                else {
                    File file = chooser.getSelectedFile();
                    FileOutputStream fileOut = new FileOutputStream(file);
                    ObjectOutputStream output = new ObjectOutputStream(
                            fileOut);
                    output.writeObject(game);
                    output.close();
                    fileOut.close();
                }
            } catch (IOException i) {
                i.printStackTrace();
            }
        }
```

```
                // GameStatus is checked last after the move has been made.
                GameStatus g = game.getGameStatus();

                if (g == GameStatus.X_WON) {
                    JOptionPane.showMessageDialog(null,
                            "X won.\nThe game will reset");
                }
                if (g == GameStatus.O_WON) {
                    JOptionPane.showMessageDialog(null,
                            "O won.\nThe game will reset");
                }
                if (g == GameStatus.CATS) {
                    JOptionPane.showMessageDialog(null,
                            "Both X and O lost.\nThe game will reset");
                }

                // Finally if the game is not in progress it is reset.
                if (g != GameStatus.IN_PROGRESS) {
                    reset();
                }
            }

        }
    }
```

Listing 4: GameStatus.java

```
/****************************************************************
 * An enumerated class with four states, X_WON, O_WON, CATS, and IN_PROGRESS.
 *
 * @author Russ Johnson
 * @version 10.10.2012
 ****************************************************************/
package package1;

public enum GameStatus {
    X_WON, O_WON, CATS, IN_PROGRESS
}
```

Listing 5: Cell.java

```
/****************************************************************
 * An enumerated class with three states, X, O, and EMPTY.
 *
 * @author Russ Johnson
 * @version 10.10.2012
 ****************************************************************/
package package1;

public enum Cell {
    X, O, EMPTY
}
```