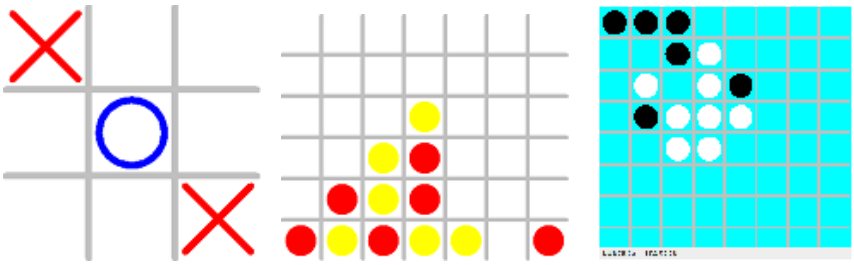
Java Graphics Programming

Case Study on Tic-Tac-Toe & Assignment

Click on the Image to run the Applet Demo



1. Let's Start with a 2-Player Console Non-OO Tic-Tac-Toe

Let us start with a 2-player console (non-graphics) version of Tic-Tac-Toe, where player `'X'` and player `'O'` enter their moves successively, as shown below:

```
Player 'X', enter your move (row[1-3] column[1-3]): 2 2
   |   |
-----------
   | X |
-----------
   |   |

Player 'O', enter your move (row[1-3] column[1-3]): 1 1
 O |   |
-----------
   | X |
-----------
   |   |

Player 'X', enter your move (row[1-3] column[1-3]): 1 3
 O |   | X
-----------
   | X |
-----------
   |   |

Player 'O', enter your move (row[1-3] column[1-3]): 3 1
 O |   | X
-----------
   | X |
-----------
 O |   |

Player 'X', enter your move (row[1-3] column[1-3]): 2 2
This move at (2,2) is not valid. Try again...
Player 'X', enter your move (row[1-3] column[1-3]): 2 3
 O |   | X
```

```
-----------
   | X | X
-----------
 O |   |

Player 'O', enter your move (row[1-3] column[1-3]): 2 1
 O |   | X
-----------
 O | X | X
-----------
 O |   |

Player 'O' won!
```

TTTCosnoleNonOO2P.java

```
 1  import java.util.Scanner;
 2  /**
 3   * Tic-Tac-Toe: Two-player console, non-graphics, non-OO version.
 4   * All variables/methods are declared as static (belong to the class)
 5   *  in the non-OO version.
 6   */
 7  public class TTTConsoleNonOO2P {
 8     // Name-constants to represent the seeds and cell contents
 9     public static final int EMPTY = 0;
10     public static final int CROSS = 1;
11     public static final int NOUGHT = 2;
12
13     // Name-constants to represent the various states of the game
14     public static final int PLAYING = 0;
15     public static final int DRAW = 1;
16     public static final int CROSS_WON = 2;
17     public static final int NOUGHT_WON = 3;
18
19     // The game board and the game status
20     public static final int ROWS = 3, COLS = 3; // number of rows and columns
21     public static int[][] board = new int[ROWS][COLS]; // game board in 2D array
22                                                        //  containing (EMPTY, CROSS, NOUGHT)
23     public static int currentState;  // the current state of the game
24                                      // (PLAYING, DRAW, CROSS_WON, NOUGHT_WON)
25     public static int currentPlayer; // the current player (CROSS or NOUGHT)
26     public static int currntRow, currentCol; // current seed's row and column
27
28     public static Scanner in = new Scanner(System.in); // the input Scanner
29
30     /** The entry main method (the program starts here) */
31     public static void main(String[] args) {
32        // Initialize the game-board and current status
33        initGame();
34        // Play the game once
35        do {
36           playerMove(currentPlayer); // update currentRow and currentCol
37           updateGame(currentPlayer, currntRow, currentCol); // update currentState
38           printBoard();
39           // Print message if game-over
40           if (currentState == CROSS_WON) {
41              System.out.println("'X' won! Bye!");
42           } else if (currentState == NOUGHT_WON) {
43              System.out.println("'O' won! Bye!");
44           } else if (currentState == DRAW) {
45              System.out.println("It's a Draw! Bye!");
46           }
47           // Switch player
48           currentPlayer = (currentPlayer == CROSS) ? NOUGHT : CROSS;
49        } while (currentState == PLAYING); // repeat if not game-over
50     }
51
52     /** Initialize the game-board contents and the current states */
53     public static void initGame() {
54        for (int row = 0; row < ROWS; ++row) {
55           for (int col = 0; col < COLS; ++col) {
56              board[row][col] = EMPTY;  // all cells empty
57           }
58        }
```

```
59          currentState = PLAYING; // ready to play
60          currentPlayer = CROSS;  // cross plays first
61       }
62
63       /** Player with the "theSeed" makes one move, with input validation.
64           Update global variables "currentRow" and "currentCol". */
65       public static void playerMove(int theSeed) {
66          boolean validInput = false;  // for input validation
67          do {
68             if (theSeed == CROSS) {
69                System.out.print("Player 'X', enter your move (row[1-3] column[1-3]): ");
70             } else {
71                System.out.print("Player 'O', enter your move (row[1-3] column[1-3]): ");
72             }
73             int row = in.nextInt() - 1;  // array index starts at 0 instead of 1
74             int col = in.nextInt() - 1;
75             if (row >= 0 && row < ROWS && col >= 0 && col < COLS && board[row][col] == EMPTY) {
76                currntRow = row;
77                currentCol = col;
78                board[currntRow][currentCol] = theSeed;  // update game-board content
79                validInput = true;  // input okay, exit loop
80             } else {
81                System.out.println("This move at (" + (row + 1) + "," + (col + 1)
82                      + ") is not valid. Try again...");
83             }
84          } while (!validInput);  // repeat until input is valid
85       }
86
87       /** Update the "currentState" after the player with "theSeed" has placed on
88           (currentRow, currentCol). */
89       public static void updateGame(int theSeed, int currentRow, int currentCol) {
90          if (hasWon(theSeed, currentRow, currentCol)) {  // check if winning move
91             currentState = (theSeed == CROSS) ? CROSS_WON : NOUGHT_WON;
92          } else if (isDraw()) {  // check for draw
93             currentState = DRAW;
94          }
95          // Otherwise, no change to currentState (still PLAYING).
96       }
97
98       /** Return true if it is a draw (no more empty cell) */
99       // TODO: Shall declare draw if no player can "possibly" win
100      public static boolean isDraw() {
101         for (int row = 0; row < ROWS; ++row) {
102            for (int col = 0; col < COLS; ++col) {
103               if (board[row][col] == EMPTY) {
104                  return false;  // an empty cell found, not draw, exit
105               }
106            }
107         }
108         return true;  // no empty cell, it's a draw
109      }
110
111      /** Return true if the player with "theSeed" has won after placing at
112          (currentRow, currentCol) */
113      public static boolean hasWon(int theSeed, int currentRow, int currentCol) {
114         return (board[currentRow][0] == theSeed         // 3-in-the-row
115                     && board[currentRow][1] == theSeed
116                     && board[currentRow][2] == theSeed
117               || board[0][currentCol] == theSeed      // 3-in-the-column
118                     && board[1][currentCol] == theSeed
119                     && board[2][currentCol] == theSeed
120               || currentRow == currentCol            // 3-in-the-diagonal
121                     && board[0][0] == theSeed
122                     && board[1][1] == theSeed
123                     && board[2][2] == theSeed
124               || currentRow + currentCol == 2  // 3-in-the-opposite-diagonal
125                     && board[0][2] == theSeed
126                     && board[1][1] == theSeed
127                     && board[2][0] == theSeed);
128      }
129
130      /** Print the game board */
131      public static void printBoard() {
132         for (int row = 0; row < ROWS; ++row) {
```

```
            for (int col = 0; col < COLS; ++col) {
               printCell(board[row][col]); // print each of the cells
               if (col != COLS - 1) {
                  System.out.print("|");   // print vertical partition
               }
            }
            System.out.println();
            if (row != ROWS - 1) {
               System.out.println("-----------"); // print horizontal partition
            }
         }
         System.out.println();
      }

      /** Print a cell with the specified "content" */
      public static void printCell(int content) {
         switch (content) {
            case EMPTY:  System.out.print("   "); break;
            case NOUGHT: System.out.print(" O "); break;
            case CROSS:  System.out.print(" X "); break;
         }
      }
   }
```

Dissecting the Program

Non-OO programs (like C programs) are organized in methods, which access common global variables. (OO programs are organized in classes.) All the variables/methods shall be declared `static` (belong to the class instead of instances). The program starts at the `main()` method. No instance will be created.

A board game (such as Tic-tac-toe) is typically programmed as a *state machine*. Depending on the current-state and the player's move, the game goes into the next-state. In this example, I use a variable `currentState` to keep track of the *current-state* of the game, and define named-constants to denote the various states of the game (`PLAYING`, `DRAW`, `CROSS_WON`, and `NOUGHT_WON`). A method called `updateGame()` is defined, which will be called after every move to update this `currentState`, by checking the status of the game-board.

Two methods are defined for printing the game board, `printBoard()` and `printCell()`. The `printBoard()` shall call `printCell()` to print each of the 9 cells. This seems trivial here, but will be useful in the object-oriented design to separate the board and cells into separate classes.

[TODO] more explanation

TRY: Prompt the user whether to play again after gameover.

```
// in main()
do {
   // Play the game once
   initGame();
   ......
   ......
   // Prompt the user whether to play again
   System.out.print("Play again (y/n)? ");
   char ans = in.next().charAt(0);
   if (ans != 'y' && ans != 'Y') {
      System.out.println("Bye!");
      System.exit(0);  // terminate the program
   }
} while (true);  // repeat until user did not answer yes
```

2. A Console OO Tic-Tac-Toe

Let us convert the earlier non-OO version of Tic-Tac-Toe to object-oriented. The OO version of this simple Tic-Tac-Toe is more complex than the non-OO version, because Tic-Tac-Toe is a rather simple application. But OO design is a necessity to build a complex application.

Enumerations `State` and `Seed`

In our earlier version, we used `int` named-constants to represent the various game states, as follows:

```
// Named-constants to represent the various states of the game
public static final int PLAYING = 0;
public static final int DRAW = 1;
public static final int CROSS_WON = 2;
public static final int NOUGHT_WON = 3;

// The current state of the game
public static int currentState = PLAYING;  // Assigned to a name, which is easier to read and understand,
                                           // instead of an int number 0
```

This approach of using `int` named-constants is better than using number in the programming statements, but it is not ideal. This is because you may inadvertently assign an `int` value *outside the valid range* to the variable `currentState`. For example,

```
currentState = 99;   // A logical error but can compile
```

JDK 1.5 introduces a new feature called *enumeration*, which is a *special class* for storing *an enumeration (list) of items*. In our case, we can define an enumeration called `GameState` as follows:

```
1  /**
2   *  Enumeration for the various states of the game
3   */
4  public enum GameState {  // to save as "GameState.java"
5     PLAYING, DRAW, CROSS_WON, NOUGHT_WON
6  }
```

To reference an *item* in an `enum`, use *`enumName.itemName`* (e.g., `GameState.PLAYING` and `GameState.DRAW`), just like referencing `static` variables of a class (e.g., `Math.PI`).

You can create an instance for an `enum` (just like creating an instance of a `class`) and assign a value into it. We shall now declare the variable `currentState` as an instance of `GameState`, which can take the value of `GameState.PLAYING`, `GameState.DRAW`, `GameState.CROSS_WON`, and `GameState.NOUGHT_WON`.

```
private GameState currentState;    // declare variable currentState as an instance of enum GameState
currentState = GameState.PLAYING;  // assign a value (an enum item) to the variable currentState
```

Take note that you can only assign a value defined in the enumeration (such as `GameState.PLAYING`, `GameState.DRAW`), and NOT an arbitrary `int` value in the earlier example. Enum is SAFE!

We shall also create an `enum` called `Seed` for the various seeds and cell contents.

```
1  /**
2   * Enumeration for the seeds and cell contents
3   */
4  public enum Seed { // to save as "Seed.java"
5     EMPTY, CROSS, NOUGHT
6  }
```

Again, you need to use `Seed.EMPTY`, `Seed.CROSS`, `Seed.NOUGHT` to refer to these values, just like any `public static` variable.

We shall declare the variables `currentPlayer` and `content` as instances of enum `Seed`.

```
private Seed currentPlayer;  // declare variable currentPlayer as an instance of Seed
currentPlayer = Seed.CROSS;  // assign a value (an enum item) to the variable currentPlayer

private Seed content;   // cell's content
content = Seed.EMPTY;
```

In brief, an `enum` is just *a special class with a list of named-constants*. But enum is safe, compared with name-constants.

Classes `Board` and `Cell`

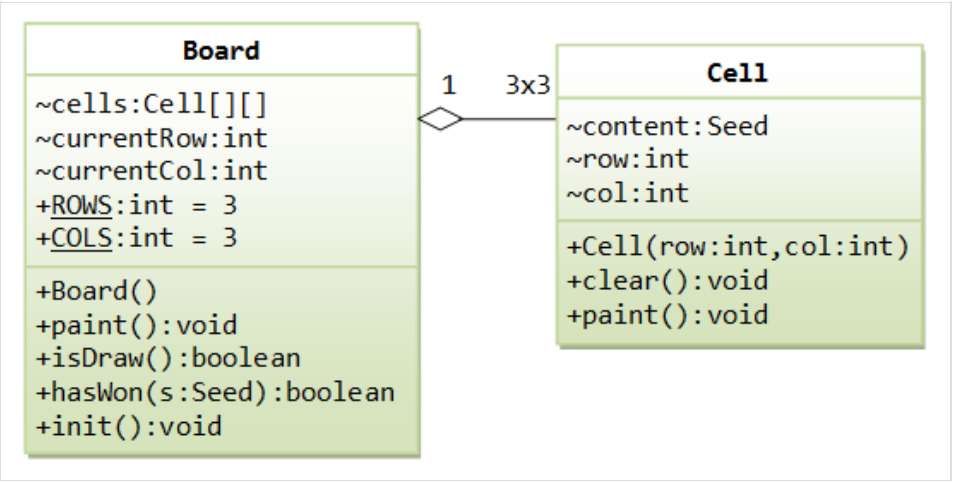
Next, let's design the OO classes needed for our Tic-Tac-Toe game. Each class shall maintain its own attributes and operations (variables and methods), and *it can paint itself* in a graphics program.

We begin with two classes, a class `Cell` for each individual cell of the game board, and a class `Board` for the 3x3 game board.

The `Cell` class has an instance variable called `content` (with package access), of the type `enum Seed`. You can only assign a value from the `enum`'s constants, such as `Seed.EMPTY`, `Seed.CROSS`, and `Seed.NOUGHT`, into `content`. A `Cell` can `paint()` itself and has its own operations such as `clear()`.

The `Board` class composes of nine `Cell` instances, arranged in an 3×3 array called `cells` (with package access), of the type `Cell[]`

[]. A Board can paint() itself, and supports its own operations such as checking the status of the current board (isDraw(), hasWon()).



Cell.java

```
/**
 * The Cell class models each individual cell of the game board.
 */
public class Cell {  // save as Cell.java
   // package access
   Seed content; // content of this cell of type Seed.
                 // take a value of Seed.EMPTY, Seed.CROSS, or Seed.NOUGHT
   int row, col; // row and column of this cell, not used in this program

   /** Constructor to initialize this cell */
   public Cell(int row, int col) {
      this.row = row;
      this.col = col;
      clear();  // clear content
   }

   /** Clear the cell content to EMPTY */
   public void clear() {
      content = Seed.EMPTY;
   }

   /** Paint itself */
   public void paint() {
      switch (content) {
         case CROSS:  System.out.print(" X "); break;
         case NOUGHT: System.out.print(" O "); break;
         case EMPTY:  System.out.print("   "); break;
      }
   }
}
```

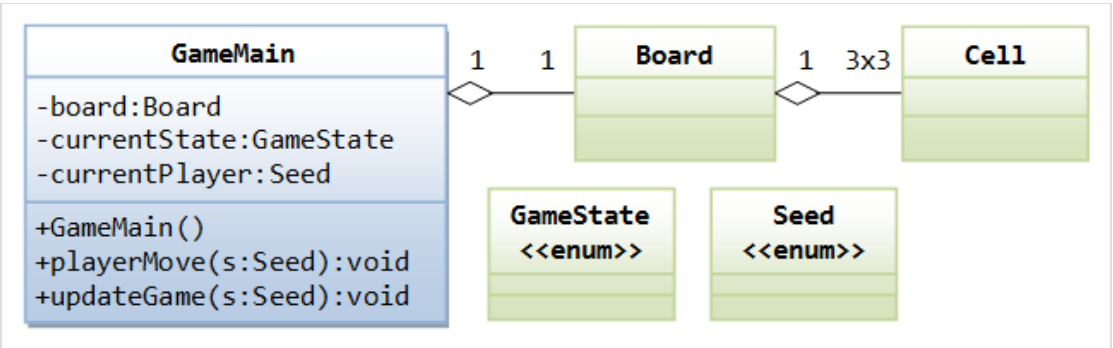
Board.java

```
/**
 * The Board class models the game-board.
 */
public class Board {  // save as Board.java
   // Named-constants for the dimensions
   public static final int ROWS = 3;
   public static final int COLS = 3;

   // package access
   Cell[][] cells;  // a board composes of ROWS-by-COLS Cell instances
   int currentRow, currentCol;  // the current seed's row and column

   /** Constructor to initialize the game board */
   public Board() {
      cells = new Cell[ROWS][COLS];  // allocate the array
      for (int row = 0; row < ROWS; ++row) {
         for (int col = 0; col < COLS; ++col) {
            cells[row][col] = new Cell(row, col); // allocate element of the array
         }
      }
   }

   /** Initialize (or re-initialize) the contents of the game board */
```

```
   public void init() {
      for (int row = 0; row < ROWS; ++row) {
         for (int col = 0; col < COLS; ++col) {
            cells[row][col].clear();  // clear the cell content
         }
      }
   }

   /** Return true if it is a draw (i.e., no more EMPTY cell) */
   public boolean isDraw() {
      for (int row = 0; row < ROWS; ++row) {
         for (int col = 0; col < COLS; ++col) {
            if (cells[row][col].content == Seed.EMPTY) {
               return false; // an empty seed found, not a draw, exit
            }
         }
      }
      return true; // no empty cell, it's a draw
   }

   /** Return true if the player with "theSeed" has won after placing at
       (currentRow, currentCol) */
   public boolean hasWon(Seed theSeed) {
      return (cells[currentRow][0].content == theSeed         // 3-in-the-row
                  && cells[currentRow][1].content == theSeed
                  && cells[currentRow][2].content == theSeed
            || cells[0][currentCol].content == theSeed      // 3-in-the-column
                  && cells[1][currentCol].content == theSeed
                  && cells[2][currentCol].content == theSeed
            || currentRow == currentCol            // 3-in-the-diagonal
                  && cells[0][0].content == theSeed
                  && cells[1][1].content == theSeed
                  && cells[2][2].content == theSeed
            || currentRow + currentCol == 2    // 3-in-the-opposite-diagonal
                  && cells[0][2].content == theSeed
                  && cells[1][1].content == theSeed
                  && cells[2][0].content == theSeed);
   }

   /** Paint itself */
   public void paint() {
      for (int row = 0; row < ROWS; ++row) {
         for (int col = 0; col < COLS; ++col) {
            cells[row][col].paint();   // each cell paints itself
            if (col < COLS - 1) System.out.print("|");
         }
         System.out.println();
         if (row < ROWS - 1) {
            System.out.println("-----------");
         }
      }
   }
}
```

Class `GameMain`

Finally, let's write a main class called `GameMain` to pull all the pieces together. `GameMain` acts as the overall *controller* for the game.



`GameMain.java`

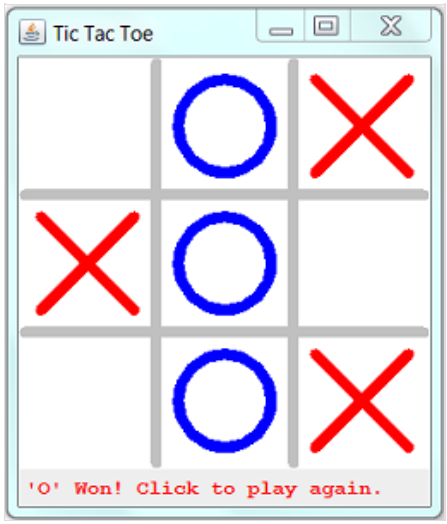
```
import java.util.Scanner;
/**
 * The main class for the Tic-Tac-Toe (Console-OO, non-graphics version)
 * It acts as the overall controller of the game.
 */
```

```
public class GameMain {
   private Board board;            // the game board
   private GameState currentState; // the current state of the game (of enum GameState)
   private Seed currentPlayer;     // the current player (of enum Seed)

   private static Scanner in = new Scanner(System.in);  // input Scanner

   /** Constructor to setup the game */
   public GameMain() {
      board = new Board();  // allocate game-board

      // Initialize the game-board and current status
      initGame();
      // Play the game once. Players CROSS and NOUGHT move alternately.
      do {
         playerMove(currentPlayer); // update the content, currentRow and currentCol
         board.paint();             // ask the board to paint itself
         updateGame(currentPlayer); // update currentState
         // Print message if game-over
         if (currentState == GameState.CROSS_WON) {
            System.out.println("'X' won! Bye!");
         } else if (currentState == GameState.NOUGHT_WON) {
            System.out.println("'O' won! Bye!");
         } else if (currentState == GameState.DRAW) {
            System.out.println("It's Draw! Bye!");
         }
         // Switch player
         currentPlayer = (currentPlayer == Seed.CROSS) ? Seed.NOUGHT : Seed.CROSS;
      } while (currentState == GameState.PLAYING);  // repeat until game-over
   }

   /** Initialize the game-board contents and the current states */
   public void initGame() {
      board.init();  // clear the board contents
      currentPlayer = Seed.CROSS;       // CROSS plays first
      currentState = GameState.PLAYING; // ready to play
   }

   /** The player with "theSeed" makes one move, with input validation.
       Update Cell's content, Board's currentRow and currentCol. */
   public void playerMove(Seed theSeed) {
      boolean validInput = false;  // for validating input
      do {
         if (theSeed == Seed.CROSS) {
            System.out.print("Player 'X', enter your move (row[1-3] column[1-3]): ");
         } else {
            System.out.print("Player 'O', enter your move (row[1-3] column[1-3]): ");
         }
         int row = in.nextInt() - 1;
         int col = in.nextInt() - 1;
         if (row >= 0 && row < Board.ROWS && col >= 0 && col < Board.COLS
               && board.cells[row][col].content == Seed.EMPTY) {
            board.cells[row][col].content = theSeed;
            board.currentRow = row;
            board.currentCol = col;
            validInput = true; // input okay, exit loop
         } else {
            System.out.println("This move at (" + (row + 1) + "," + (col + 1)
                  + ") is not valid. Try again...");
         }
      } while (!validInput);   // repeat until input is valid
   }

   /** Update the currentState after the player with "theSeed" has moved */
   public void updateGame(Seed theSeed) {
      if (board.hasWon(theSeed)) {  // check for win
         currentState = (theSeed == Seed.CROSS) ? GameState.CROSS_WON : GameState.NOUGHT_WON;
      } else if (board.isDraw()) {  // check for draw
         currentState = GameState.DRAW;
      }
      // Otherwise, no change to current state (still GameState.PLAYING).
   }

   /** The entry main() method */
```

```
80      public static void main(String[] args) {
81         new GameMain();  // Let the constructor do the job
82      }
83   }
```

Take note that the OO-version and the non-OO version have the same codes, but are organized differently. The organization in OO enables you to design and develop complex system.

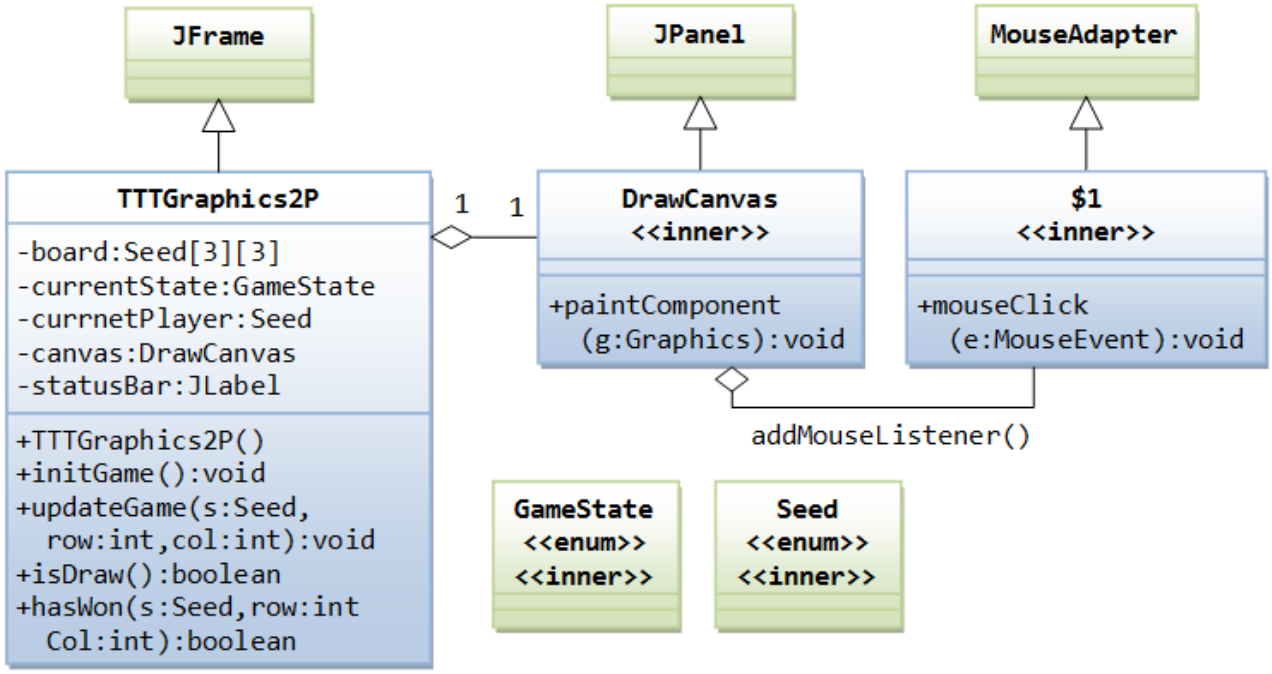
3. A Graphics Simple-OO Tic-Tac-Toe



Let's rewrite the "console" version into a "graphics" version - a Java Swing application, as illustrated. In this initial design, we do not separate the cell and board into dedicated classes, but include them in the main class. We used an inner class `DrawCanvas` (that extends `JPanel`) to do the custom drawing, and an anonymous inner class for `MouseListener`.

The content-pane (of the top-level container `JFrame`) is set to `BorderLayout`. The `DrawCanvas` (`JPanel`) is placed at the `CENTER`; while a status-bar (a `JLabel`) is placed at the `SOUTH` (`PAGE_END`).

The class diagram is as follows:



TTTGraphics2P.java

```
1   import java.awt.*;
2   import java.awt.event.*;
3   import javax.swing.*;
4   /**
5    * Tic-Tac-Toe: Two-player Graphics version with Simple-OO
6    */
7   @SuppressWarnings("serial")
8   public class TTTGraphics2P extends JFrame {
9      // Named-constants for the game board
10     public static final int ROWS = 3;  // ROWS by COLS cells
11     public static final int COLS = 3;
12
13     // Named-constants of the various dimensions used for graphics drawing
14     public static final int CELL_SIZE = 100; // cell width and height (square)
15     public static final int CANVAS_WIDTH = CELL_SIZE * COLS;  // the drawing canvas
```

```
16      public static final int CANVAS_HEIGHT = CELL_SIZE * ROWS;
17      public static final int GRID_WIDTH = 8;                   // Grid-line's width
18      public static final int GRID_WIDHT_HALF = GRID_WIDTH / 2; // Grid-line's half-width
19      // Symbols (cross/nought) are displayed inside a cell, with padding from border
20      public static final int CELL_PADDING = CELL_SIZE / 6;
21      public static final int SYMBOL_SIZE = CELL_SIZE - CELL_PADDING * 2; // width/height
22      public static final int SYMBOL_STROKE_WIDTH = 8; // pen's stroke width
23
24      // Use an enumeration (inner class) to represent the various states of the game
25      public enum GameState {
26         PLAYING, DRAW, CROSS_WON, NOUGHT_WON
27      }
28      private GameState currentState;  // the current game state
29
30      // Use an enumeration (inner class) to represent the seeds and cell contents
31      public enum Seed {
32         EMPTY, CROSS, NOUGHT
33      }
34      private Seed currentPlayer;  // the current player
35
36      private Seed[][] board   ; // Game board of ROWS-by-COLS cells
37      private DrawCanvas canvas; // Drawing canvas (JPanel) for the game board
38      private JLabel statusBar;  // Status Bar
39
40      /** Constructor to setup the game and the GUI components */
41      public TTTGraphics2P() {
42         canvas = new DrawCanvas();  // Construct a drawing canvas (a JPanel)
43         canvas.setPreferredSize(new Dimension(CANVAS_WIDTH, CANVAS_HEIGHT));
44
45         // The canvas (JPanel) fires a MouseEvent upon mouse-click
46         canvas.addMouseListener(new MouseAdapter() {
47            @Override
48            public void mouseClicked(MouseEvent e) {  // mouse-clicked handler
49               int mouseX = e.getX();
50               int mouseY = e.getY();
51               // Get the row and column clicked
52               int rowSelected = mouseY / CELL_SIZE;
53               int colSelected = mouseX / CELL_SIZE;
54
55               if (currentState == GameState.PLAYING) {
56                  if (rowSelected >= 0 && rowSelected < ROWS && colSelected >= 0
57                        && colSelected < COLS && board[rowSelected][colSelected] == Seed.EMPTY) {
58                     board[rowSelected][colSelected] = currentPlayer; // Make a move
59                     updateGame(currentPlayer, rowSelected, colSelected); // update state
60                     // Switch player
61                     currentPlayer = (currentPlayer == Seed.CROSS) ? Seed.NOUGHT : Seed.CROSS;
62                  }
63               } else {       // game over
64                  initGame(); // restart the game
65               }
66               // Refresh the drawing canvas
67               repaint();  // Call-back paintComponent().
68            }
69         });
70
71         // Setup the status bar (JLabel) to display status message
72         statusBar = new JLabel("  ");
73         statusBar.setFont(new Font(Font.DIALOG_INPUT, Font.BOLD, 15));
74         statusBar.setBorder(BorderFactory.createEmptyBorder(2, 5, 4, 5));
75
76         Container cp = getContentPane();
77         cp.setLayout(new BorderLayout());
78         cp.add(canvas, BorderLayout.CENTER);
79         cp.add(statusBar, BorderLayout.PAGE_END); // same as SOUTH
80
81         setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
82         pack();  // pack all the components in this JFrame
83         setTitle("Tic Tac Toe");
84         setVisible(true);  // show this JFrame
85
86         board = new Seed[ROWS][COLS]; // allocate array
87         initGame(); // initialize the game board contents and game variables
88      }
89
```

```
90      /** Initialize the game-board contents and the status */
91      public void initGame() {
92         for (int row = 0; row < ROWS; ++row) {
93            for (int col = 0; col < COLS; ++col) {
94               board[row][col] = Seed.EMPTY; // all cells empty
95            }
96         }
97         currentState = GameState.PLAYING; // ready to play
98         currentPlayer = Seed.CROSS;       // cross plays first
99      }
100
101     /** Update the currentState after the player with "theSeed" has placed on
102         (rowSelected, colSelected). */
103     public void updateGame(Seed theSeed, int rowSelected, int colSelected) {
104        if (hasWon(theSeed, rowSelected, colSelected)) {  // check for win
105           currentState = (theSeed == Seed.CROSS) ? GameState.CROSS_WON : GameState.NOUGHT_WON;
106        } else if (isDraw()) {  // check for draw
107           currentState = GameState.DRAW;
108        }
109        // Otherwise, no change to current state (still GameState.PLAYING).
110     }
111
112     /** Return true if it is a draw (i.e., no more empty cell) */
113     public boolean isDraw() {
114        for (int row = 0; row < ROWS; ++row) {
115           for (int col = 0; col < COLS; ++col) {
116              if (board[row][col] == Seed.EMPTY) {
117                 return false; // an empty cell found, not draw, exit
118              }
119           }
120        }
121        return true;  // no more empty cell, it's a draw
122     }
123
124     /** Return true if the player with "theSeed" has won after placing at
125         (rowSelected, colSelected) */
126     public boolean hasWon(Seed theSeed, int rowSelected, int colSelected) {
127        return (board[rowSelected][0] == theSeed  // 3-in-the-row
128              && board[rowSelected][1] == theSeed
129              && board[rowSelected][2] == theSeed
130           || board[0][colSelected] == theSeed      // 3-in-the-column
131              && board[1][colSelected] == theSeed
132              && board[2][colSelected] == theSeed
133           || rowSelected == colSelected            // 3-in-the-diagonal
134              && board[0][0] == theSeed
135              && board[1][1] == theSeed
136              && board[2][2] == theSeed
137           || rowSelected + colSelected == 2  // 3-in-the-opposite-diagonal
138              && board[0][2] == theSeed
139              && board[1][1] == theSeed
140              && board[2][0] == theSeed);
141     }
142
143     /**
144      *  Inner class DrawCanvas (extends JPanel) used for custom graphics drawing.
145      */
146     class DrawCanvas extends JPanel {
147        @Override
148        public void paintComponent(Graphics g) {  // invoke via repaint()
149           super.paintComponent(g);    // fill background
150           setBackground(Color.WHITE); // set its background color
151
152           // Draw the grid-lines
153           g.setColor(Color.LIGHT_GRAY);
154           for (int row = 1; row < ROWS; ++row) {
155              g.fillRoundRect(0, CELL_SIZE * row - GRID_WIDHT_HALF,
156                    CANVAS_WIDTH-1, GRID_WIDTH, GRID_WIDTH, GRID_WIDTH);
157           }
158           for (int col = 1; col < COLS; ++col) {
159              g.fillRoundRect(CELL_SIZE * col - GRID_WIDHT_HALF, 0,
160                    GRID_WIDTH, CANVAS_HEIGHT-1, GRID_WIDTH, GRID_WIDTH);
161           }
162
163           // Draw the Seeds of all the cells if they are not empty
```

```
164            // Use Graphics2D which allows us to set the pen's stroke
165            Graphics2D g2d = (Graphics2D)g;
166            g2d.setStroke(new BasicStroke(SYMBOL_STROKE_WIDTH, BasicStroke.CAP_ROUND,
167                  BasicStroke.JOIN_ROUND));  // Graphics2D only
168            for (int row = 0; row < ROWS; ++row) {
169               for (int col = 0; col < COLS; ++col) {
170                  int x1 = col * CELL_SIZE + CELL_PADDING;
171                  int y1 = row * CELL_SIZE + CELL_PADDING;
172                  if (board[row][col] == Seed.CROSS) {
173                     g2d.setColor(Color.RED);
174                     int x2 = (col + 1) * CELL_SIZE - CELL_PADDING;
175                     int y2 = (row + 1) * CELL_SIZE - CELL_PADDING;
176                     g2d.drawLine(x1, y1, x2, y2);
177                     g2d.drawLine(x2, y1, x1, y2);
178                  } else if (board[row][col] == Seed.NOUGHT) {
179                     g2d.setColor(Color.BLUE);
180                     g2d.drawOval(x1, y1, SYMBOL_SIZE, SYMBOL_SIZE);
181                  }
182               }
183            }
184
185            // Print status-bar message
186            if (currentState == GameState.PLAYING) {
187               statusBar.setForeground(Color.BLACK);
188               if (currentPlayer == Seed.CROSS) {
189                  statusBar.setText("X's Turn");
190               } else {
191                  statusBar.setText("O's Turn");
192               }
193            } else if (currentState == GameState.DRAW) {
194               statusBar.setForeground(Color.RED);
195               statusBar.setText("It's a Draw! Click to play again.");
196            } else if (currentState == GameState.CROSS_WON) {
197               statusBar.setForeground(Color.RED);
198               statusBar.setText("'X' Won! Click to play again.");
199            } else if (currentState == GameState.NOUGHT_WON) {
200               statusBar.setForeground(Color.RED);
201               statusBar.setText("'O' Won! Click to play again.");
202            }
203         }
204      }
205
206      /** The entry main() method */
207      public static void main(String[] args) {
208         // Run GUI codes in the Event-Dispatching thread for thread safety
209         SwingUtilities.invokeLater(new Runnable() {
210            @Override
211            public void run() {
212               new TTTGraphics2P(); // Let the constructor do the job
213            }
214         });
215      }
216   }
```

Dissecting the Program

[TODO]

4. Game Programming Assignment

You can use the above Tic-tac-toe as a template to develop board games such as Connect-4 and Othello.

4.1 Connect-Four

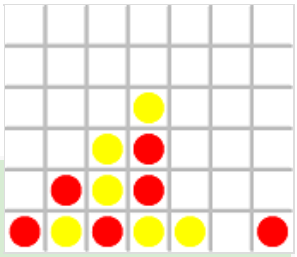
Click on the image to run the demo (in applet).

Wiki "Connect-4" to understand the rules of the game.

To write a Connect-Four game, let's start from Tic-Tac-Toe's "Graphics Version". Do the following changes on "`TTTGraphics2P.java`":

1. Change constants `ROWS` to 6 and `COLS` to 7. Run the program. You shall see a 6×7 grid. Try clicking on the cells, "cross" and

"nought" shall be displayed alternately.



2. Modify the `mouseClicked()` event-handler to position the seed at the "bottom" row of the column clicked, instead of on the the cell clicked. You need to check that there is empty cell on that column.

```
if (colSelected >= 0 && colSelected < COLS) {
   // Look for an empty cell starting from the bottom row
   for (int row = ROWS -1; row >= 0; row--) {
      if (board[row][colSelected] == Seed.EMPTY) {
         board[row][colSelected] = currentPlayer; // Make a move
         updateGame(currentPlayer, row, colSelected); // update state
         // Switch player
         currentPlayer = (currentPlayer == Seed.CROSS) ? Seed.NOUGHT : Seed.CROSS;
         break;
      }
   }
}
```

3. Modify the `hasWon()` method to check for 4-in-a-line (along row, column, diagonal or opposite-diagonal).

```
// HINTS:
public boolean hasWon(Seed theSeed, int rowSelected, int colSelected) {
   // Check for 4-in-a-line on the rowSelected
   int count = 0;
   for (int col = 0; col < COLS; ++col) {
      if (board[rowSelected][col] == theSeed) {
         ++count;
         if (count == 4) return true;  // found
      } else {
         count = 0; // reset and count again if not consecutive
      }
   }
   // Check column and diagonals
   ......
   return false;  // no 4-in-a-line found
}
```

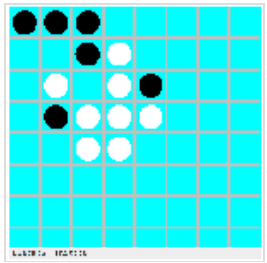
That's all!

Next,

1. Tidy up the names (In Eclipse, Refactor ⇒ Rename).
2. Tidy up the display (using red and yellow discs, instead of cross and nought).
3. Add more features. For example, sound effect; or buttons to control the game.
4. Re-design your classes (Read the "Graphics Advanced-OO Tic-Tac-Toe").
5. Improve your display (e.g., using images, animation etc).

4.2 Othello (Reversi)

Click on the image to run my demo (in applet).



Wiki "Othello" or "Reversi" to understand the rules of the game.

Modify the above Tic-Tac-Toe ("`TTTGraphics2P.java`"):

1. Change `ROWS` and `COLS` to 8. Run the program. You shall see a 8×8 grid. Try clicking on the cells, "cross" and "nought" shall be displayed alternately.
2. Modify the `updateGame(Seed theSeed, int rowSelected, int colSelect)` to flip the opponent's seeds along the row, column, diagonal and opposite diagonal - centered at `(rowSelected, colSelected)` - after the player with "`theSeed`" has placed on `(rowSelected, colSelected)`. If thre is no more empty space, the game is over. Decide the winner by counting the numbers of black and white seeds.
   HINTS:

```
public void updateGame(Seed mySeed, int rowSelected, int colSelected) {
   Seed opponentSeed = (mySeed == Seed.BLACK) ? Seed.WHITE : Seed.BLACK;
   int col, row;

   // Flip opponent's seeds along the row to the right if any
   col = colSelected + 1;
   // Look for adjacent opponent's seeds up to 2nd last column
```

```
   while (col < COLS - 1 && board[rowSelected][col] == opponentSeed) {
      ++col;
   }
   // Look for my seed immediately after opponent's seeds
   if (col <= COLS - 1 && board[rowSelected][col] == mySeed) {
      // Flip opponent's seeds in between to my seeds
      for (int colFlip = colSelected + 1; colFlip <= col - 1; ++colFlip) {
         board[rowSelected][colFlip] = mySeed;
      }
   }
   ......
   // Check for game over and declare winner
   ......
```

3. Remove `isDraw()` and `hasWon()`.

Next,

1. Tidy up the names (Refactor ⇒ Rename).
2. Tidy up the display (using black and white discs, instead of cross and nought).
3. Add more features. For example, sound effect; or buttons to control the game.
4. Re-design your classes (Read the "Graphics Advanced-OO Tic-Tac-Toe").
5. Improve your display (e.g., using images, animation etc).

4.3 Sudoku

You could wiki "Sudoku" to understand the rules of the game.

Sudoku's graphics does not involve custom drawing (such as drawing lines or circles). Hence, the above Tic-Tac-Toe graphics example is not really applicable. You can simply use a 9x9 `JTextFields` arranged in a 9x9 `GridLayout` - the GUI codes is simple!

The steps for producing the display are:

- Set the `JFrame`'s content-pane to 9×9 `GridLayout`. Create 9×9 `JTextFields` and add to the content-pane. You need to set up two 9×9 arrays. One `int[9][9]` to store the numbers (1-9, or 0 if empty). Another `JTextField[9][9]` to do the display (print blank if the number is 0).
- Initialize the game by reading in an input puzzle with blank cells, and populate the `int[9][9]` and `JTextField[9][9]` arrays. Set the non-empty cells to non-editable.

For example,

```
1  import java.awt.*;
2  import java.awt.event.*;
3  import javax.swing.*;
4
5  public class SudokuDisplayOnly extends JFrame {
6     // Name-constants for the various dimensions
7     public static final int ROWS = 9; // ROWS by COLS cells
8     public static final int COLS = 9;
9     public static final int CELL_SIZE = 60; // Cell width/height
10    public static final int CANVAS_WIDTH = CELL_SIZE * COLS;
11    public static final int CANVAS_HEIGHT = CELL_SIZE * ROWS;
12
13    // Game board
14    private int[][] cells;
15    private JTextField[][] tfCells;
16
17    // Puzzle to solve. Can have more and pick one in random.
18    private int[][] puzzle =
19       {{5, 3, 4, 6, 7, 8, 9, 1, 2},
20        {6, 7, 2, 1, 9, 5, 3, 4, 8},
21        {1, 9, 8, 3, 4, 2, 5, 6, 7},
22        {8, 5, 9, 7, 6, 1, 4, 2, 3},
23        {4, 2, 6, 8, 5, 3, 7, 9, 1},
24        {7, 1, 3, 9, 2, 4, 8, 5, 6},
25        {9, 6, 1, 5, 3, 7, 2, 8, 4},
26        {2, 8, 7, 4, 1, 9, 6, 3, 5},
27        {3, 4, 5, 2, 8, 6, 1, 7, 9}};
28
```

```
    // Mask for puzzle should be generated randomly
    private boolean[][] mask =
       {{false, false, false, false, false, true, false, false, false},
        {false, false, false, false, false, false, false, false, true},
        {false, false, true, false, false, false, false, false, false},
        {true, false, false, false, false, false, false, false, false},
        {false, false, false, true, false, false, false, false, false},
        {false, false, false, false, false, false, true, false, false},
        {false, false, false, false, false, false, false, true, false},
        {false, true, false, false, false, false, false, false, false},
        {false, false, false, false, true, false, false, false, false}};

    /** Constructor to setup the game and the GUI */
    public SudokuDisplayOnly() {
       Container cp = getContentPane();
       cp.setLayout(new GridLayout(ROWS, COLS));

       cells = new int[ROWS][COLS];
       tfCells = new JTextField[ROWS][COLS]; // allocate JTextField array

       // Create 9x9 JTextFields and place on the GridLayout
       for (int row = 0; row < ROWS; ++row) {
          for (int col = 0; col < COLS; ++col) {
             tfCells[row][col] = new JTextField(); // allocate element of array
             cp.add(tfCells[row][col]);  // ContentPane adds JTextField
             int number = puzzle[row][col];
             if (mask[row][col]) {
                cells[row][col] = 0;
                tfCells[row][col].setText("");  // empty
                tfCells[row][col].setEditable(true);
                tfCells[row][col].setBackground(Color.YELLOW);
             } else {
                cells[row][col] = number;
                tfCells[row][col].setText(number + "");
                tfCells[row][col].setEditable(false);
             }
             tfCells[row][col].setHorizontalAlignment(JTextField.CENTER);
             tfCells[row][col].setFont(new Font("Monospaced", Font.BOLD, 20));
          }
       }
       cp.setPreferredSize(new Dimension(CANVAS_WIDTH, CANVAS_HEIGHT));

       setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
       pack();
       setTitle("Sudoku");
       setVisible(true);
    }

    /** The entry main() entry method */
    public static void main(String[] args) {
       // Run the GUI construction on the event-dispatching thread for thread safety
       javax.swing.SwingUtilities.invokeLater(new Runnable() {
          @Override
          public void run() {
             new SudokuDisplayOnly(); // Let the constructor do the job
          }
       });
    }
 }
```

Some useful methods of `JTextField` are:

```
setBackground(Color c)  // Set the background color of the component
setForeground(Color c)  // Set the text color of the JTextField
setFont(Font f)         // Set the font used by the JTextField
setHorizontalAlignment(int align);  // align: JTextField.CENTER, JTextField.LEFT, JTextField.RIGHT
```

Next, write the event handler `actionPerformed()` for the `ActionEvent` fired by the `JTextField`. You may use one listener to listen to all the 9×9 `JTextField`s. In order to ascertain the `JTextField` that has fired the `ActionEvent`. You could use the `event.getSource()` method to retrieve the source object that has fired the event and compare with all the 9×9 `JTextField`s:

```
@Override
public void actionPerformed(ActionEvent evt) {
   int rowSelected = -1;
```

```
   int colSelected = -1;
   // Get the source object that fired the event
   JTextField source = (JTextField)e.getSource();
   boolean found = false;
   for (int row = 0; row < ROWS && !found; ++row) {
      for (int col = 0; col < COLS && !found; ++col) {
         if (tfCells[row][col] == source) {
            rowSelected = row;
            colSelected = col;
            found = true;
         }
      }
   }
   ......
```

More:

- Validate the input (show invalid input in a different color or show it in "X") and check for puzzle solved.
- Refine your display. Use 3x3 `JPanel`s in `GridLayout`. Each `JPanel` has 3x3 `JTextField`s in `GridLayout` too. In this way, you can control the border of the `JPanel`s via `setBorder()`.
- Add more features (e.g., sound effect, buttons for controlling the game, undo).
- Improve the game, e.g., difficulty level (easy, medium, hard), hints and cheats, etc.
- Re-design your classes.
- Improve display (e.g., images and animation).

Triggering `JTextField`'s `ActionEvent` involves hitting the "enter" key. That is, without hitting the enter key, the number is not captured by `actionPerformed()`, although it may appear on the text field. Try using the `KeyEvent`, which is fired after every key stroke.

4.4 Mine Sweeper

Similar to Sudoku, the graphics for Mine Sweeper does not involve custom drawings. For the basic version with 10x10 cells, construct a 10x10 `JButton` array and arranged in `GridLayout`. Study the sample code in Sudoku to create the display.

In Mine Sweeper, you need to response to left-mouse click and right-mouse-click differently. Hence, instead of listening to the `ActionEvent`, you shall listen to the `MouseEvent` with mouse-clicked handler so as to response to the left-click and right-click. Since `ActionEvent` is not used, you probably can use 10x10 `JLabel` instead of `JButton`, as `JLabel` can also trigger mouse-event.

4.5 MasterMind

[TODO]

4.6 Checker

[TODO]

5. A Graphics Advanced-OO Tic-Tac-Toe - Separating the `Board` and `Cell` Classes

5.1 A More Versatile Swing Program Template

Before I proceed, I have modified the Swing program template to make the codes more versatile (the same codes can be run as a standalone application, as well as Applet and Java Web-Start application).

1. The main class extends from `JPanel` instead of `JFrame`.
2. In the entry `main()` method, an instance of `JFrame` is constructed, and its content-pane is set to an instance of the main class. The `JFrame` is then set visible.

```
1  import java.awt.*;
2  import java.awt.event.*;
3  import javax.swing.*;
4
5  /** ... Purpose of this program ... */
```

```
@SuppressWarnings("serial")
public class SwingTemplateJPanel extends JPanel {
   // Name-constants
   public static final int CANVAS_WIDTH = 640;
   public static final int CANVAS_HEIGHT = 480;
   public static final String TITLE = "...Title...";
   // ......

   // Decalare private variables of GUI components
   // ......

   /** Constructor to setup the GUI components */
   public SwingTemplateJPanel() {
      setPreferredSize(new Dimension(CANVAS_WIDTH, CANVAS_HEIGHT));
      // "this" JPanel container sets layout
      // setLayout(new ....Layout());

      // Allocate the GUI components
      // .....

      // "this" JPanel adds components
      // add(....)

      // Source objects add listeners
      // .....
   }

   /** Custom painting codes on this JPanel */
   @Override
   public void paintComponent(Graphics g) {
      super.paintComponent(g);  // fill background
      setBackground(Color.BLACK);

      // Custom painting codes
      // ......
   }

   /** The entry main() method */
   public static void main(String[] args) {
      // Run GUI codes in the Event-Dispatching thread for thread safety
      SwingUtilities.invokeLater(new Runnable() {
         public void run() {
            JFrame frame = new JFrame(TITLE);
            // Set the content-pane of the JFrame to an instance of main JPanel
            frame.setContentPane(new SwingTemplateJPanel());
            frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            frame.pack();             // "this" JFrame packs its components
            frame.setLocationRelativeTo(null); // center the application window
            frame.setVisible(true);            // show it
         }
      });
   }
}
```

Running as a Standalone Application

The above template has a `main()` method, which can be run as a standalone application, a web-start application, or from a JAR file.

Running as an Applet

An Swing Applet extends `javax.swing.JApplet`, intead of `javax.swing.JFrame`. An applet starts at `init()`, instead of `main()`. We can provide another class to run the main class as an applet as follows:

```
import javax.swing.*;

/** Applet for .... */
@SuppressWarnings("serial")
public class SwingTemplateJApplet extends JApplet {

   /** init() to setup the UI components */
   @Override
   public void init() {
      // Run GUI codes in the Event-Dispatching thread for thread safety
      try {
```

```
            // Use invokeAndWait() to ensure that init() exits after GUI construction
            SwingUtilities.invokeAndWait(new Runnable() {
               @Override
               public void run() {
                  // Set the content-pane of "this" JApplet to an instance of main JPanel
                  setContentPane(new SwingTemplateJPanel());
               }
            });
         } catch (Exception e) {
            e.printStackTrace();
         }
      }
   }
```

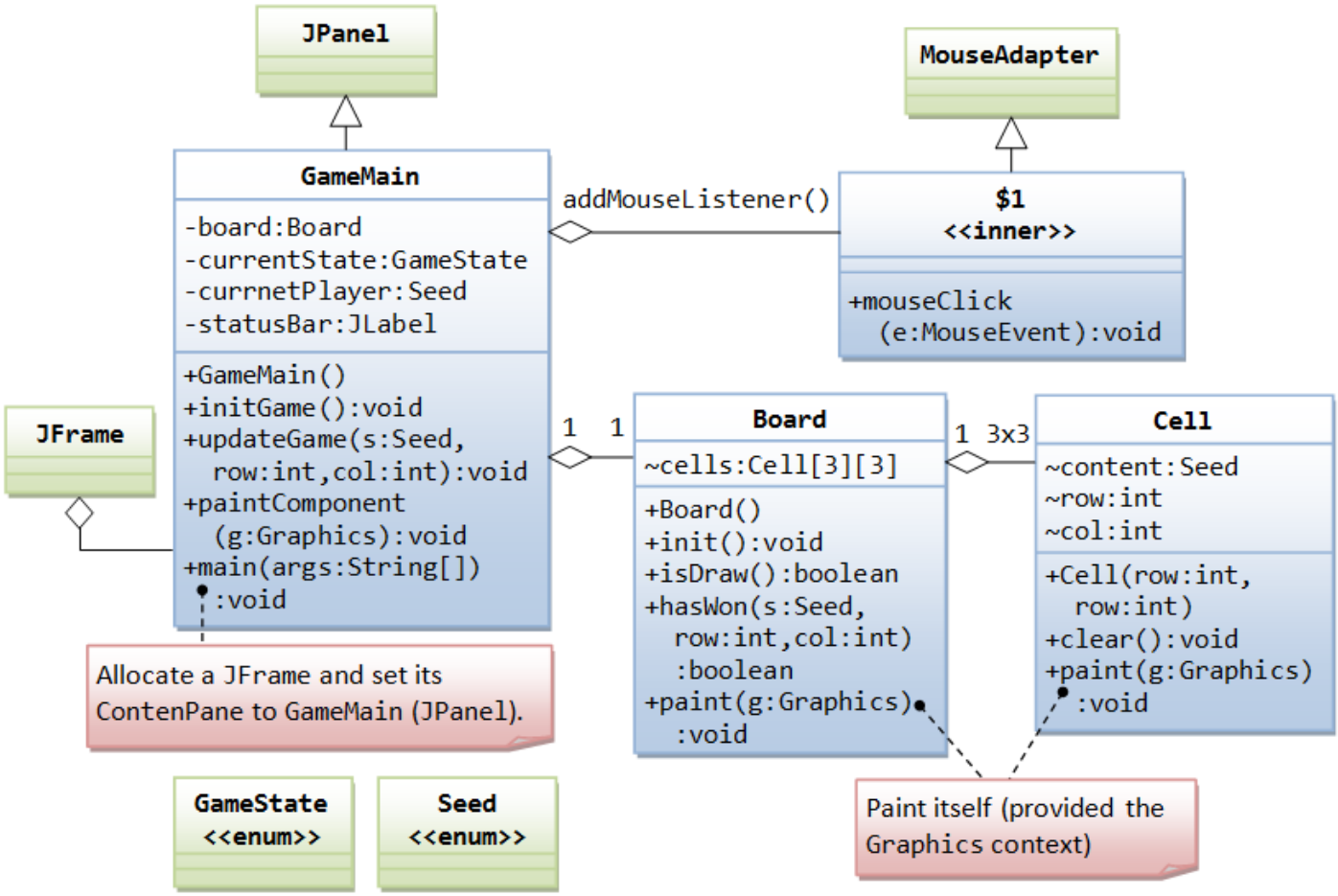
You need to provide an HMTL file to run the applet in production. (For testing, You could run your applet directly under Eclipse/NetBeans using the so-called "appletviewer" without an HTML file.) For example,

```
<html>
<head>
  <title>An .... Applet</title>
</head>
<body>
  <h1>Heading ...</h1>
  <applet code="SwingTemplateJApplet.class" width="640" height="480" alt="Error Loading Applet?!">
  Your browser does not seem to support &lt;APPLET&gt; tag!
  </applet>
</body>
</html>
```

5.2 Graphics with OO Design

In a good OO design, each class shall be encapsulated, shall have its own attributes and operations (variables and methods), and responsible for *painting itself* in a graphics program.

The class diagram is as follows:



Enumeration `Seed.java`

```
/**
 * Enumeration for the seeds and cell contents
```

```
 */
public enum Seed {  // to save as "Seed.java"
   EMPTY, CROSS, NOUGHT
}
```

Enumeration `State.java`

```
/**
 *  Enumeration for the various states of the game
 */
public enum State {  // to save as "GameState.java"
   PLAYING, DRAW, CROSS_WON, NOUGHT_WON
}
```

Class `Cell.java`

```
import java.awt.*;
/**
 * The Cell class models each individual cell of the game board.
 */
public class Cell {
   // Package access
   Seed content; // content of this cell (Seed.EMPTY, Seed.CROSS, or Seed.NOUGHT)
   int row, col; // row and column of this cell

   /** Constructor to initialize this cell with the specified row and col */
   public Cell(int row, int col) {
      this.row = row;
      this.col = col;
      clear(); // clear content
   }

   /** Clear this cell's content to EMPTY */
   public void clear() {
      content = Seed.EMPTY;
   }

   /** Paint itself on the graphics canvas, given the Graphics context */
   public void paint(Graphics g) {
      // Use Graphics2D which allows us to set the pen's stroke
      Graphics2D g2d = (Graphics2D)g;
      g2d.setStroke(new BasicStroke(GameMain.SYMBOL_STROKE_WIDTH,
            BasicStroke.CAP_ROUND, BasicStroke.JOIN_ROUND)); // Graphics2D only
      // Draw the Seed if it is not empty
      int x1 = col * GameMain.CELL_SIZE + GameMain.CELL_PADDING;
      int y1 = row * GameMain.CELL_SIZE + GameMain.CELL_PADDING;
      if (content == Seed.CROSS) {
         g2d.setColor(Color.RED);
         int x2 = (col + 1) * GameMain.CELL_SIZE - GameMain.CELL_PADDING;
         int y2 = (row + 1) * GameMain.CELL_SIZE - GameMain.CELL_PADDING;
         g2d.drawLine(x1, y1, x2, y2);
         g2d.drawLine(x2, y1, x1, y2);
      } else if (content == Seed.NOUGHT) {
         g2d.setColor(Color.BLUE);
         g2d.drawOval(x1, y1, GameMain.SYMBOL_SIZE, GameMain.SYMBOL_SIZE);
      }
   }
}
```

Class `Board.java`

```
import java.awt.*;
/**
 * The Board class models the ROWS-by-COLS game-board.
 */
public class Board {
   // package access
   Cell[][] cells; // composes of 2D array of ROWS-by-COLS Cell instances

   /** Constructor to initialize the game board */
   public Board() {
      cells = new Cell[GameMain.ROWS][GameMain.COLS]; // allocate the array
      for (int row = 0; row < GameMain.ROWS; ++row) {
         for (int col = 0; col < GameMain.COLS; ++col) {
```

```
            cells[row][col] = new Cell(row, col); // allocate element of array
         }
      }
   }

   /** Initialize (or re-initialize) the game board */
   public void init() {
      for (int row = 0; row < GameMain.ROWS; ++row) {
         for (int col = 0; col < GameMain.COLS; ++col) {
            cells[row][col].clear(); // clear the cell content
         }
      }
   }

   /** Return true if it is a draw (i.e., no more EMPTY cell) */
   public boolean isDraw() {
      for (int row = 0; row < GameMain.ROWS; ++row) {
         for (int col = 0; col < GameMain.COLS; ++col) {
            if (cells[row][col].content == Seed.EMPTY) {
               return false; // an empty seed found, not a draw, exit
            }
         }
      }
      return true; // no empty cell, it's a draw
   }

   /** Return true if the player with "seed" has won after placing at
       (seedRow, seedCol) */
   public boolean hasWon(Seed seed, int seedRow, int seedCol) {
      return (cells[seedRow][0].content == seed   // 3-in-the-row
                  && cells[seedRow][1].content == seed
                  && cells[seedRow][2].content == seed
               || cells[0][seedCol].content == seed // 3-in-the-column
                  && cells[1][seedCol].content == seed
                  && cells[2][seedCol].content == seed
               || seedRow == seedCol               // 3-in-the-diagonal
                  && cells[0][0].content == seed
                  && cells[1][1].content == seed
                  && cells[2][2].content == seed
               || seedRow + seedCol == 2           // 3-in-the-opposite-diagonal
                  && cells[0][2].content == seed
                  && cells[1][1].content == seed
                  && cells[2][0].content == seed);
   }

   /** Paint itself on the graphics canvas, given the Graphics context */
   public void paint(Graphics g) {
      // Draw the grid-lines
      g.setColor(Color.GRAY);
      for (int row = 1; row < GameMain.ROWS; ++row) {
         g.fillRoundRect(0, GameMain.CELL_SIZE * row - GameMain.GRID_WIDHT_HALF,
               GameMain.CANVAS_WIDTH - 1, GameMain.GRID_WIDTH,
               GameMain.GRID_WIDTH, GameMain.GRID_WIDTH);
      }
      for (int col = 1; col < GameMain.COLS; ++col) {
         g.fillRoundRect(GameMain.CELL_SIZE * col - GameMain.GRID_WIDHT_HALF, 0,
               GameMain.GRID_WIDTH, GameMain.CANVAS_HEIGHT - 1,
               GameMain.GRID_WIDTH, GameMain.GRID_WIDTH);
      }

      // Draw all the cells
      for (int row = 0; row < GameMain.ROWS; ++row) {
         for (int col = 0; col < GameMain.COLS; ++col) {
            cells[row][col].paint(g);  // ask the cell to paint itself
         }
      }
   }
}
```

Classes `GameMain.java`

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
```

```java
/**
 * Tic-Tac-Toe: Two-player Graphic version with better OO design.
 * The Board and Cell classes are separated in their own classes.
 */
@SuppressWarnings("serial")
public class GameMain extends JPanel {
   // Named-constants for the game board
   public static final int ROWS = 3;  // ROWS by COLS cells
   public static final int COLS = 3;
   public static final String TITLE = "Tic Tac Toe";

   // Name-constants for the various dimensions used for graphics drawing
   public static final int CELL_SIZE = 100; // cell width and height (square)
   public static final int CANVAS_WIDTH = CELL_SIZE * COLS;  // the drawing canvas
   public static final int CANVAS_HEIGHT = CELL_SIZE * ROWS;
   public static final int GRID_WIDTH = 8;  // Grid-line's width
   public static final int GRID_WIDHT_HALF = GRID_WIDTH / 2; // Grid-line's half-width
   // Symbols (cross/nought) are displayed inside a cell, with padding from border
   public static final int CELL_PADDING = CELL_SIZE / 6;
   public static final int SYMBOL_SIZE = CELL_SIZE - CELL_PADDING * 2;
   public static final int SYMBOL_STROKE_WIDTH = 8; // pen's stroke width

   private Board board;            // the game board
   private GameState currentState; // the current state of the game
   private Seed currentPlayer;     // the current player
   private JLabel statusBar;       // for displaying status message

   /** Constructor to setup the UI and game components */
   public GameMain() {

      // This JPanel fires MouseEvent
      this.addMouseListener(new MouseAdapter() {
         @Override
         public void mouseClicked(MouseEvent e) {  // mouse-clicked handler
            int mouseX = e.getX();
            int mouseY = e.getY();
            // Get the row and column clicked
            int rowSelected = mouseY / CELL_SIZE;
            int colSelected = mouseX / CELL_SIZE;

            if (currentState == GameState.PLAYING) {
               if (rowSelected >= 0 && rowSelected < ROWS
                     && colSelected >= 0 && colSelected < COLS
                     && board.cells[rowSelected][colSelected].content == Seed.EMPTY) {
                  board.cells[rowSelected][colSelected].content = currentPlayer; // move
                  updateGame(currentPlayer, rowSelected, colSelected); // update currentState
                  // Switch player
                  currentPlayer = (currentPlayer == Seed.CROSS) ? Seed.NOUGHT : Seed.CROSS;
               }
            } else {        // game over
               initGame();  // restart the game
            }
            // Refresh the drawing canvas
            repaint();  // Call-back paintComponent().
         }
      });

      // Setup the status bar (JLabel) to display status message
      statusBar = new JLabel("         ");
      statusBar.setFont(new Font(Font.DIALOG_INPUT, Font.BOLD, 14));
      statusBar.setBorder(BorderFactory.createEmptyBorder(2, 5, 4, 5));
      statusBar.setOpaque(true);
      statusBar.setBackground(Color.LIGHT_GRAY);

      setLayout(new BorderLayout());
      add(statusBar, BorderLayout.PAGE_END); // same as SOUTH
      setPreferredSize(new Dimension(CANVAS_WIDTH, CANVAS_HEIGHT + 30));
            // account for statusBar in height

      board = new Board();   // allocate the game-board
      initGame();  // Initialize the game variables
   }

   /** Initialize the game-board contents and the current-state */
```

```
78      public void initGame() {
79         for (int row = 0; row < ROWS; ++row) {
80            for (int col = 0; col < COLS; ++col) {
81               board.cells[row][col].content = Seed.EMPTY; // all cells empty
82            }
83         }
84         currentState = GameState.PLAYING;  // ready to play
85         currentPlayer = Seed.CROSS;        // cross plays first
86      }
87
88      /** Update the currentState after the player with "theSeed" has placed on (row, col) */
89      public void updateGame(Seed theSeed, int row, int col) {
90         if (board.hasWon(theSeed, row, col)) {  // check for win
91            currentState = (theSeed == Seed.CROSS) ? GameState.CROSS_WON : GameState.NOUGHT_WON;
92         } else if (board.isDraw()) {  // check for draw
93            currentState = GameState.DRAW;
94         }
95         // Otherwise, no change to current state (PLAYING).
96      }
97
98      /** Custom painting codes on this JPanel */
99      @Override
100     public void paintComponent(Graphics g) {  // invoke via repaint()
101        super.paintComponent(g);    // fill background
102        setBackground(Color.WHITE); // set its background color
103
104        board.paint(g);  // ask the game board to paint itself
105
106        // Print status-bar message
107        if (currentState == GameState.PLAYING) {
108           statusBar.setForeground(Color.BLACK);
109           if (currentPlayer == Seed.CROSS) {
110              statusBar.setText("X's Turn");
111           } else {
112              statusBar.setText("O's Turn");
113           }
114        } else if (currentState == GameState.DRAW) {
115           statusBar.setForeground(Color.RED);
116           statusBar.setText("It's a Draw! Click to play again.");
117        } else if (currentState == GameState.CROSS_WON) {
118           statusBar.setForeground(Color.RED);
119           statusBar.setText("'X' Won! Click to play again.");
120        } else if (currentState == GameState.NOUGHT_WON) {
121           statusBar.setForeground(Color.RED);
122           statusBar.setText("'O' Won! Click to play again.");
123        }
124     }
125
126     /** The entry "main" method */
127     public static void main(String[] args) {
128        // Run GUI construction codes in Event-Dispatching thread for thread safety
129        javax.swing.SwingUtilities.invokeLater(new Runnable() {
130           public void run() {
131              JFrame frame = new JFrame(TITLE);
132              // Set the content-pane of the JFrame to an instance of main JPanel
133              frame.setContentPane(new GameMain());
134              frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
135              frame.pack();
136              frame.setLocationRelativeTo(null); // center the application window
137              frame.setVisible(true);            // show it
138           }
139        });
140     }
141  }
```

5.3 Running as a Standalone Program

Simply run the class containing the entry `main()` method.

5.4 Deploying an Application via a JAR file

To deploy an application containing many classes, you have to pack (i.e., jar) all classes and resources into a single file, with a

manifest that specifies the main class (containing the entry `main()` method).

For example:

- via the Eclipse's "Export" option: Right-click on the project ⇒ Export ⇒ Java ⇒ JAR file ⇒ Next ⇒ Specify the JAR filename ⇒ Next ⇒ Next ⇒ Select "Generate the manifest file" ⇒ Browse to select the main class "`GameMain`" ⇒ Finish.
- via the "`jar`" command.
  First, create a manifest file called "`tictactoe.mf`", as follow:

  ```
  Manifest-Version: 1.0
  Main-Class: GameMain
  ```

  Next, issue a "`jar`" command (form CMD shell) where options `'c'` for create, `'m'` for manifest, `'f'` for output jar filename, and `'v'` for verbose:

  ```
  > jar cmfv tictactoe.mf tictactoe.jar *.class
  ```

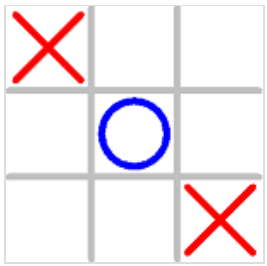
You can run the program from a JAR file directly (without unpacking the JAR file) by:

1. In Windows' Explorer, right-click on the JAR file ⇒ Open with ⇒ Java Platform SE Binary; or
2. From the CMD shell, run `java.exe` with `-jar` option, i.e., "`java -jar JarFileName.jar`".

Note: JAR file uses the ZIP algorithm. In other words, you could use WinZIP/WinRAR to open and extract the contents of a JAR file.

5.5 Running as an Applet

Click the image to run the demo applet:



`AppletMain.java`

Provide a main class (says `AppletMain.java`) for the applet that extends `javax.swing.JApplet`:

```
1  import javax.swing.*;
2
3  /** Tic-tac-toe Applet */
4  @SuppressWarnings("serial")
5  public class AppletMain extends JApplet {
6
7     /** init() to setup the GUI components */
8     @Override
9     public void init() {
10       // Run GUI codes in the Event-Dispatching thread for thread safety
11       try {
12          // Use invokeAndWait() to ensure that init() exits after GUI construction
13          SwingUtilities.invokeAndWait(new Runnable() {
14             @Override
15             public void run() {
16                setContentPane(new GameMain());
17             }
18          });
19       } catch (Exception e) {
20          e.printStackTrace();
21       }
22    }
23 }
```

`TicTacToe.html`

Provide an HMTL file (says "`TicTacToe.html`") that embeds the "`AppletMain.class`":

```
1  <html>
2  <head>
3    <title>Tic Tac Toe</title>
4  </head>
5  <body>
6    <h1>Tic Tac Toe</h1>
7    <applet code="AppletMain.class" width="300" height="330" alt="Error Loading Applet?!">
8    Your browser does not seem to support &lt;APPLET&gt; tag!
```

```
  </applet>
</body>
</html>
```

tictactoe.jar

To deploy an applet which contains more than one classes, you need to pack all the classes and resources into a JAR file (e.g., via Eclipse's "Export" option or "`jar`" command described earlier), but you need not use a manifest (for specify a main class as applet does not need a `main()` method). Then, use the following `<applet>` tag with an "`archive`" attribute to specify the JAR filename:

```
<applet code="AppletMain.class"
   archive="JarFileName.jar"
   width="300" height="300"
   alt="Error Loading Applet?!" >
Your browser does not seem to support &lt;APPLET&gt; tag!
</applet>
```

6. Adding Sound Effect

6.1 How to Play an Audio File

SoundTest.java

```
import java.awt.*;
import java.awt.event.*;
import java.net.URL;
import javax.sound.sampled.*;
import javax.swing.*;

/** Test playing sound file */
@SuppressWarnings("serial")
public class SoundTest extends JFrame {
   private String fileGameOver = "gameover.wav"; // audio filename ("wav", "au", "aiff" only)
   private Clip soundClipGameOver;               // Java Internal Sound Clip

   /** Constructor to setup the GUI components and sound clips */
   public SoundTest() {
      // Prepare the Sound Clip
      try {
         // Generate an URL from filename
         URL url = this.getClass().getClassLoader().getResource(fileGameOver);
         if (url == null) {
            System.err.println("Couldn't find file: " + fileGameOver);
         } else {
            // Get an audio input stream to read the audio data
            AudioInputStream audioIn = AudioSystem.getAudioInputStream(url);
            // Allocate a sound clip, used by Java internal
            soundClipGameOver = AudioSystem.getClip();
            // Read the audio data into sound clip
            soundClipGameOver.open(audioIn);
         }
      } catch (UnsupportedAudioFileException e) {
         System.err.println("Audio Format not supported: " + fileGameOver);
      } catch (Exception e) {
         e.printStackTrace();
      }

      Container cp = getContentPane();
      cp.setLayout(new FlowLayout());
      JButton btn = new JButton("Play Sound");
      cp.add(btn);
      btn.addActionListener(new ActionListener() {
         @Override
         public void actionPerformed(ActionEvent e) {
            // Play sound clip
            if (soundClipGameOver.isRunning()) soundClipGameOver.stop();
            soundClipGameOver.setFramePosition(0); // rewind to the beginning
            soundClipGameOver.start();             // start playing
         }
      });
```

```
48          setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
49          setTitle("Test Sound");
50          setSize(200, 100);
51          setVisible(true);
52       }
53
54       /** The entry "main" method */
55       public static void main(String[] args) {
56          // Run GUI codes in Event-Dispatching thread for thread safety
57          SwingUtilities.invokeLater(new Runnable() {
58             @Override
59             public void run() {
60                new SoundTest(); // Let the constructor do the job
61             }
62          });
63       }
64    }
```

- Prepare a sound file called "`gameover.wav`" (try googling) and place it under under your project root (or "`bin`") directory.
- If you receive error message "Couldn't find file: xxx". Try moving your file around your project (such as under the "bin" sub-directory).
- JDK supports only a sampled audio format, "`.wav`", "`.au`", and "`.aiff`". It does not support "`.mp3`" (You will get an error message "Audio Format not supported: xxx". There used to be a "Java Media Framework (JMF)" that supports MP3.
- You may try to download the pronunciations for the words "game" and "over", and join them into a "`wav`" file.
- We need to test the sound effect under a Swing application, instead of placing all the codes under the `main()`. This is because `main()` exits before the sound gets a chance to play.

6.2 Adding Sound Effect to the Tic-Tac-Toe Graphics Version

Two sound clips were used in the demo: one for the move ("`move.wav`") and the other for game-over ("`gameover.wav`"). (Google to find some interesting "`wav`" files.)

Include the following codes at the *appropriate* locations:

```
// Add the following import statements
import java.io.IOException;
import java.net.URL;
import javax.sound.sampled.*;

......

// Declare the following variables for the sound clips in the main class
String fileMove = "sounds/move.wav";         // audio filename for move effect
String fileGameOver = "sounds/gameover.wav"; // audio filename for game-over effect
Clip soundClipMove;      // Sound clip for move effect
Clip soundClipGameOver;  // Sound clip for game-over effect

......

// In the main class's Constructor, prepare the sound clips
try {
   URL url = this.getClass().getClassLoader().getResource(fileGameOver);
   if (url == null) {
      System.err.println("Couldn't find file: " + fileGameOver);
   } else {
      AudioInputStream audioIn = AudioSystem.getAudioInputStream(url);
      soundClipGameOver = AudioSystem.getClip();
      soundClipGameOver.open(audioIn);
   }

   url = this.getClass().getClassLoader().getResource(fileMove);
   if (url == null) {
      System.err.println("Couldn't find file: " + fileMove);
   } else {
      AudioInputStream audioIn = AudioSystem.getAudioInputStream(url);
      soundClipMove = AudioSystem.getClip();
      soundClipMove.open(audioIn);
   }
} catch (UnsupportedAudioFileException e) {
   System.err.println("Audio Format not supported!");
```

```
} catch (Exception e) {
   e.printStackTrace();
}

......

// In mouseClicked() event-handler - play sound effect upon mouse-click
if (currentState == GameState.PLAYING) {
   if (soundClipMove.isRunning()) soundClipMove.stop();
   soundClipMove.setFramePosition(0); // rewind to the beginning
   soundClipMove.start();             // Start playing
} else {
   if (soundClipGameOver.isRunning()) soundClipGameOver.stop();
   soundClipGameOver.setFramePosition(0); // rewind to the beginning
   soundClipGameOver.start();             // Start playing
}
```

You may consider using different sound files for "win", "loss", "draw", "valid_move" and "invalid_move".

7. Using Images

Read "Drawing Images" of "Custom Graphics".

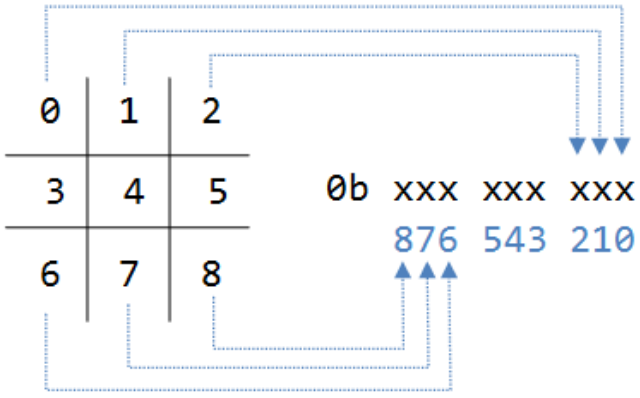
8. Animation

Read "Animation" of "Custom Graphics".

9. Fast Matching of Winning Patterns with Bit-Masks (Advanced)

Reference: Arthur van Hoff's Tic Tac Toe Applet Demo (under the JDK demo "applets" folder).

A much more efficient method for matching with a winning pattern in a Tic-tac-toe is to use *a 9-bit binary number* (stored as an `int` or `short` type) to denote the placement of the seeds, and use bit operations to perform the matching.



The following table summaries all the bit-wise operations, which are efficient and fast.

Operator	Description	Usage	Example
&	Bit-wise AND	*expr1* & *expr2*	0b0110 0001 & Ob1110 0000 gives 0b0110 0000
|	Bit-wise OR	*expr1* | *expr2*	0b0110 0001 | Ob0000 1000 gives 0b0110 1001
!	Bit-wise NOT	!*expr*	^0b0110 0001 gives 0b1001 1110
^	Bit-wise XOR	*expr1* ^ *expr2*	0b0110 0001 ^ Ob0000 0001 gives 0b0110 1001
<<	Left-shift and padded with zeros	operand << number	0b0000 0001 << 4 gives 0b0001 0000
>>	Right-shift and padded with	operand >>	0b1000 0001 >> 2 gives 0b1110

	the "sign-bit" (Signed-extended right-shift)	`number`	`0000`
>>>	Right-shift and padded with zeros (Unsigned-extended right-shift)	`operand >>> number`	`0b1000 0001 >>> 2` gives `0b0010 0000`

We can keep the 8 winning patterns in an `int` array as follows:

```
int[] winningPatterns = {
      0x1c0,   // 0b111 000 000 (row 2)
      0x038,   // 0b000 111 000 (row 1)
      0x007,   // 0b000 000 111 (row 0)
      0x124,   // 0b100 100 100 (col 2)
      0x092,   // 0b010 010 010 (col 1)
      0x049,   // 0b001 001 001 (col 0)
      0x111,   // 0b100 010 001 (diagonal)
      0x054};  // 0b001 010 100 (opposite diagonal)
   // msb is (2, 2); lsb is (0, 0)
```

Note: JDK 1.7 supports binary literals beginning with prefix "`0b`". Pre-JDK 1.7 does not support binary literals but supports hexadecimal literals beginning with prefix "`0x`". Eclipse IDE suports JDK 1.7 only after Eclipse 3.7.2. Hence, try `0b...` but fall back to `0x...` if compilation fails.

We define two placement binary patterns for the cross and nought respectively.

```
int crossPattern;
int noughtPattern;

// updating the pattern after each move
int bitPosition = rowSelected * ROWS + colSelected;
if (currentPlayer == Seed.CROSS) {
   crossPattern = crossPattern | (0x1 << bitPosition);
} else {
   noughtPattern = noughtPattern | (0x1 << bitPosition);
}
```

`(0x1 << bitPosition)` shifts a binary `0b 000 000 001` to the left by the `bitPosition` number of bits, so as to place a `'1'` bit at the proper position. It is then bit-OR with the existing pattern to include the new bit, without modifying the existing bits. For example, suppose `rowSelect = 2` and `colSelected = 0`, then `bitPosition = 6`. `(0x1 << bitPosition)` gives `0b 001 000 000`.

To match with the winning patterns:

```
public boolean hasWon(Seed theSeed) {
   int playerPattern = (theSeed == Seed.CROSS) ? crossPattern : noughtPattern;
   for (int aWinningPattern : winningPatterns) {
      if ((aWinningPattern & playerPattern) == aWinningPattern) {
         return true;
      }
   }
   return false;
}
```

`(aWinningPattern & playerPattern)` masks out all the bits in the `playerPattern` except those having 1's in `aWinningPattern`. For example, suppose that `playerPattern = 0b111 000 101`, it matches the `aWinningPattern = 0b111 000 000`. This is becasue `(playerPattern & aWinningPattern)` returns `0b111 000 000`, which is the same the the `aWinningPattern`.

This code is very much more efficient as it involves only comparison with 8 integers (plus 8 efficient bit-AND operations).

10. Other Modes of Operation

10.1 WebStart Application

[TODO]

10.2 Playing Over the Net

[TODO]

11. Playing Against Computer with AI (Advanced)

Read "Case Study on Tic-Tac-Toe Part 2: With AI".

REFERENCES & RESOURCES

1. JDK Applets demo "TicTacToe" (under JDK demo applets folder).

Latest version tested: JDK 1.7.0_17
Last modified: April, 2013