Problem Solving Session

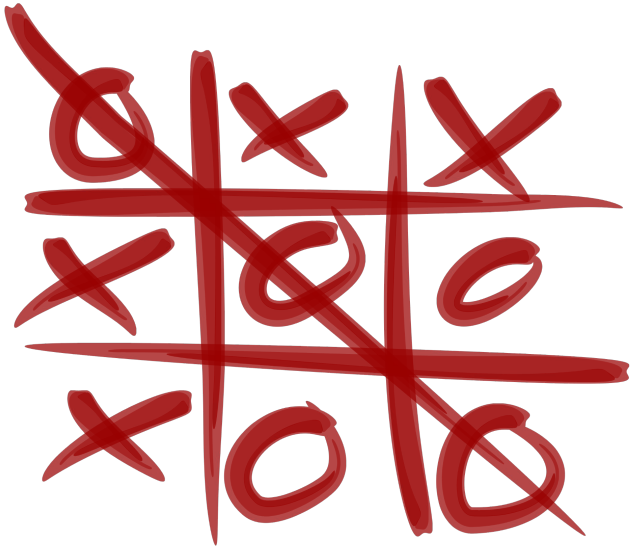
- The remainder of today's class will comprise the ***problem solving session*** (***PSS***).
- Your instructor will divide you into ***teams of 3 or 4 students***.
- Each team will ***work together*** to solve the following problems over the course of ***20-30 minutes***.
  - You may work on paper, a white board, or digitally as determined by your instructor.
  - You will submit your solution by pushing it to GitHub before the end of class.
- Your instructor will go over the solution before the end of class.
- If there is any time remaining, you will begin work on your homework assignment.

Class participation is a significant part of your grade (20%). This includes in class activities and the problem solving session.

Your Course Assistants will grade your participation by verifying that you pushed your solutions before the end of the class period each day.

Tic Tac Toe

- ***Tic Tac Toe*** is a two player game played on a board comprising 3 rows and 3 columns of squares.
- Each player chooses a symbol (traditionally Xs and Os) to make moves.
- Players take turns moving in empty squares.
- The game ends when one player makes a move such that they can connect 3 of their symbols vertically, horizontally, or diagonally on the board.
- Most games end in a stalemate - all possible moves have been made and no player has 3-in-a-row.



You will be implementing a JavaFX GUI that can be used to play a game of Tic Tac Toe.

Problem Solving 1

- Use the `TicTacToeCLI` to start a game and play.
- Take a few minutes to examine the Tic Tac Toe ***model*** and ***command line interface*** that have been provided to you, including:
  - `TicTacToeException`
  - `Move`
  - `TicTacToeStatus`
  - `TicTacToe`
  - `TicTacToeCLI`
- Answer the following questions:

How are moves made?

It uses "move" method given parameters for a row and column. And moves the symbol into the square

What happens when an invalid move is made?

It catches an error and prints out a message

How does the CLI know which player's turn it is?

The x player goes first. Each time a move is suffessfully made, it is the other player's turn.

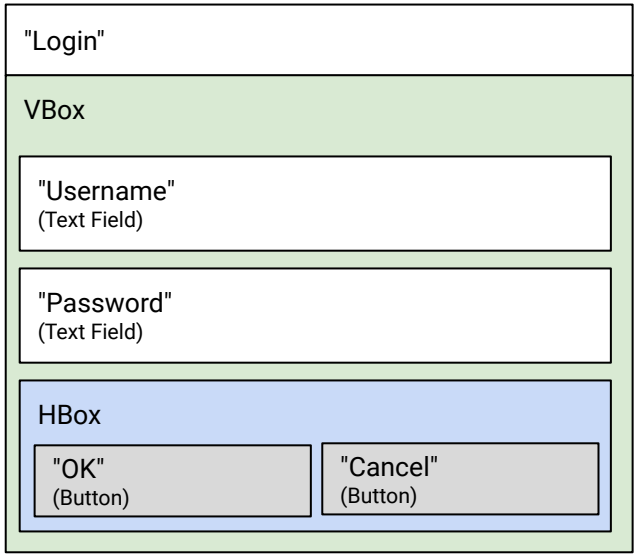
```
>> move 1 1
Game status: ONGOING
  0 1 2
0  | |
1  |X|
2  | |

>> move 2 2
Game status: ONGOING
  0 1 2
0  | |
1  |X|
2  | |O

>>
```

Problem Solving 2

Using the example to the right, work with your team to design a rough layout for your Tic Tac Toe GUI ***on paper, a whiteboard, or the slide that follows this one***.

Try to indicate the type of JavaFX node used, i.e. layouts, buttons, etc. Your GUI must include at least the following features:

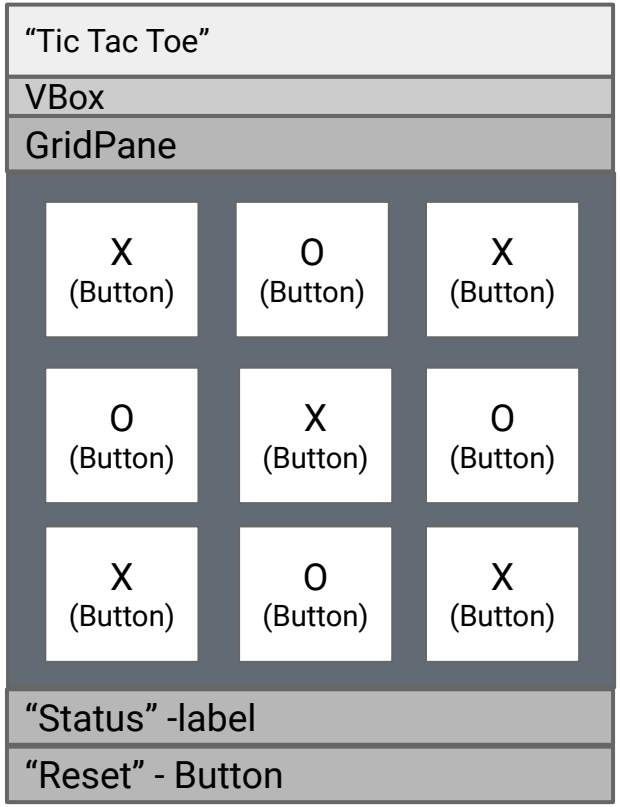
- Tic Tac Toe is played with a ***3x3 board***.
- Some ***means to make a move*** by selecting a square on the board.
- A place to display a ***status message***, e.g. feedback after a move.
- There must be a means to ***restart*** the game.



An example of a simple dialog to log in with a username and password. Color is only used to more easily tell the different controls apart.

Use this example as a reference when designing your Tic Tac Toe GUI on paper, a whiteboard, or the next slide.

Problem Solving 2: GUI Design



```
Private Button makeButton(int row, int col){
     Button button = new Button();
     button.setMaxSize(Double.MAX_VALUE,
          Double.Max_VALUE);
     button.setGraphic(new ImageView(NONE));
     button.setPadding(new Insets(0));

     button.setOnAction(new MoveMaker(ttt,
     row,col, statusLabel));

     Return button;
}
```

Problem Solving 3

Write a ***factory method*** that will create and return the JavaFX control that a player will use to represent a square on the board. Do not worry about event handling at this time.

You may assume that you have an EventHandler that observes the control and makes moves.

Consider:

- How will you know which square was selected?
- What happens is the move is invalid?

Be as detailed as you can, including any customization.

Problem Solving 4

Like Reversi, you will need to upgrade the TicTacToe model to implement an Observer Pattern so that your GUI can update when moves are made on the board.

Unlike Reversi, there is no `Square` class that can be observed directly. Instead, the ***subject*** will be the `TicTacToe` game itself.

In the space to the top right, define an interface for an observer that will be notified when a move has been made at a specific location on the board, e.g. an X being played at location *(0, 1)*.

In the space to the bottom right, write an event handler that will update the JavaFX control in your GUI that represents one of the squares on the board. You may assume that any variables that you need are in scope.

```
// observer interface
Public interface TicTacToeOberver {
      public void boardChanged(int row, int col);
}
```

```
// event handler
@Override
Public void boardChange(int row, int col){
      Try{
            Move move = ttt.getMoveAt(row,col);
            Image image = move == Move.NONE ? N :
move == Move.X ? X : 0;
      )
}
```