# IT 2045C Computer Programming II Prof. Tom Wulf Lab 07B: Software Engineering Camp Part B: Tic Tac Toe Spring 2022 10 pts

This is the second in a series of sub-assignments in which we will practice the software engineering process presented here.

For each project we will:

* Derive a candidate class list from, requirements documents, Use-Cases. And subject matter knowledge
* Create CRC cards for the candidate classes and cull the unneeded classes
* Create a UML diagram showing the relationships between the classes in our design
* Implement our classes and create JUnit tests for them. (Note that we may not create the complete applications here for each case study and then we only stub out the methods in the classes to create a test first Agile software engineering approach.

## Part B: OOP Tic Tac Toe

Previously, we coded a Swing GUI version of Tic Tac Toe by grafting a GUI onto console code that was developed in CP I. From an engineering perspective this was pretty ugly. We ended up putting most of the game logic code into the ActionListener for the Tic Tac Toe button.  
What I want you to do here is to completely re-engineer TicTacToe using the Software Engineering approach we have been exploring.

1. Using the requirements documents and general background business knowledge, we create a list of candidate classes for our application. Here for Tic Tac Toe, we are pretty familiar with the internals so consider:  
   1. Do we need a Game class?
   2. Do we need a Board class i.e. TTTBoard? Consider that we have a bunch of static methods that all use the Board. These would belong to a Board object…
   3. Should we subclass JFrame or give the Game a JFrame instance?
   4. You should use the sub-classed TTTTileButton from Ass 01!

**Put your list here with candidate class names one per line.**

* Game
* TTTBoard
* Player
* TTTTileButton
* JFrame

1. Now, create the CRC cards for each class noting the class name, class responsibilities, and any collaborators. At this point, you can cull the classes from the candidate list that you do not use. LEAVE THEM IN THE LIST BUT USE THE STRIKETHROUGH FONT TO SHOW THEY ARE ELIMINATED.  
     
   **Put the CRC cards here for the classes:**

**Game:**

* 1. Responsibilities: Manage game flow, handle turns, determine game state.
  2. Collaborators: TTTBoard, Player

**TTTBoard:**

* 1. Responsibilities: Manage board state, check for wins, update with moves.
  2. Collaborators: Game

**Player:**

* 1. Responsibilities: Represent player, track symbol (X or O).
  2. Collaborators: Game

**JFrame:**

* 1. Responsibilities: Provide main GUI window, handle events.
  2. Collaborators: Game, TTTBoard

1. Now create a complete UML diagram that shows all the classes that you did not cull and their relationships.   
     
   **Insert the UML Diagram here.**

**+-----------------+ +-------------+**

**| TTTBoard |<---------| Game |**

**+-----------------+ +-------------+**

**^ ^**

**| |**

**| |**

**+ +**

**+-----------------+ +-------------+**

**| Player | | JFrame |**

**+-----------------+ +-------------+**

**--------------------------------------------**

**| Game |**

**|------------------------------------------|**

**| - board: TTTBoard |**

**| - currentPlayer: Player |**

**|-------------------------------------------|**

**| + startGame() |**

**| + switchPlayer() |**

**| + isGameOver(): boolean |**

**| + checkWin(): boolean |**

**| + isBoardFull(): boolean |**

**| + markBoard(row: int, col: int): boolean|**

**----------------------------------------------------------**

**| ^**

**| |**

**| |**

**v |**

**----------------------------------------------------**

**| TTTBoard |**

**|--------------------------------------------------|**

**| - boardState: char[][] |**

**|-------------------------------------------------|**

**| + checkWin(): boolean |**

**| + isBoardFull(): boolean |**

**| + markBoard(row: int, col: int, symbol: char): boolean|**

**-----------------------------------------**

**| ^**

**| |**

**| |**

**v |**

**-----------------------------------------------**

**| Player |**

**|---------------------------------------------|**

**| - name: String |**

**| - symbol: char |**

**|----------------------------------------------|**

**| + getName(): String |**

**| + getSymbol(): char |**

**-----------------------------------------------**

**| ^**

**| |**

**| |**

**v |**

**-----------------------------------------**

**| TTTTileButton |**

**|---------------------------------------|**

**| - row: int |**

**| - col: int |**

**|-------------------------------------------------|**

**| + setSymbol(symbol: char): void |**

**---------------------------------------**

1. Now if you created UML Use-cases include those here:

(Not Needed for Part B)

1. Implement Tic Tac Toe as a Java Swing GUI Project using the new objects that you developed. It should be identical in the interface to the original TTT but now has a better engineered basis.  
   **Project Name: Lab07B\_OOPTTT**

A screenshot of a computer

Description automatically generated

Use the Canvas Assignment mechanism to submit the link to your GitHub repo for this project and after inserting your screen and code shots this file renamed **LastnameFirstname \_Lab\_07B.docx** (using your actual name).