OOD - Object Oriented Design

In **OOD** we focus on modeling the **objects** in a problem, the **behavior** of these objects as they interact, and the **data** that they must contain. In C++ we implement behavior as **methods** (**member functions)** and data as **public** and **private** **variables**.

***Tic-Tac-Toe***

Tic-Tac-Toe is a simple 2-player game that has a 3 x 3 board. There are two symbols used on the board, X and O, that represent the positions of the opponents. Player alternately place a single symbol on the board until one or the other has a "Win" -- three X's or O's in a row, horizontally, vertically or diagonally; or a "Draw" (aka "a cat's game") exists with neither player having a Win.

From this description we can make the following observations:

1. Tic-Tac-Toe is a game;
2. The game has a 3 x 3 board;
3. Each cell of the board may contain (has a): empty, X, O;
4. At any time the WinState of the board may be (has a): Playing, Win-X, Win-O, Draw;
5. There must be a mechanism (method) for asking for moves and recording them (i.e., a routine that "plays the game).

***Implementation***

Now we may design a class that implements our observations. Note that the "is a" relationship **→** a class, and the "has a" relationship **→** data or function within the class.

***Battleship***

We can extend our notion to the game of Battleship -- a simple 2-player game that has a 5 x 5 board. The first player randomly places 4-cell long battleship on the board, hidden from view, in either horizontal or vertical orientation. The second player "fires" missiles at the board, attempting to sink the battleship. Hits are recorded as X's, misses as O's. One keeps track of the number of missiles fired, the number of hits, and the number of the game (several games usually are played). The battleship is sunk when missiles have struck all its cells.

From this description we can make the following observations:

1. **BattleshipGame** is a game;
2. The game has a 5 x 5 board;
3. The board has a battleship randomly placed on it;
4. Each cell of the board may contain (has a): empty, X (a hit), O (a miss);
5. The game has statistics that it must track (number of missiles fired, hits, game number);
6. There must be mechanisms for asking for missile coordinates, recording them, asking if the battleship is hit or sunk (has data);
7. There must be a mechanism for displaying the current state of the game.
8. A **Battleship** is a ship;
9. A battleship has 4 cell coordinates;
10. Each battleship cell may be (has a state) not-hit or hit;
11. A battleship has a means to tell you if a coordinate hits it;
12. A battleship has a means to tell you if it is sunk or afloat.

***Implementation***

Now we may design a class that implements our observations. Again, note that the "is a" relationship **→** a class, and the "has a" relationship **→** data or function within the class.