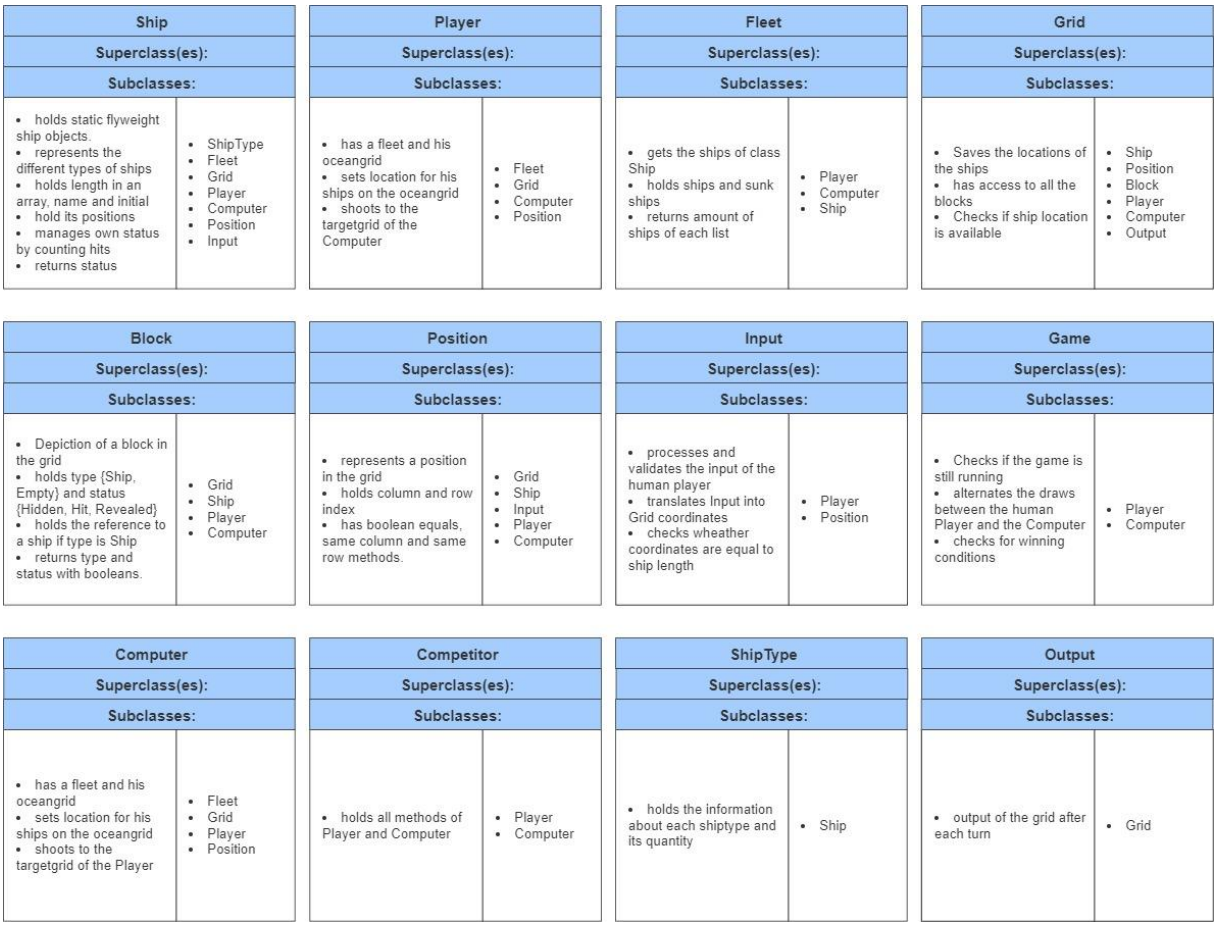
**Exercice 1**

1. **Following the Responsibility Driven Design, start from the game's requirements and rules and derive classes, responsibilities, and collaborations (use CRC cards). Describe each step you make and store the final cards in your answer.**

   Firstly, we need to develop the core elements from the game's requirements. As one of the key elements we recognize the ships. We think the best way of implementation will be to create objects for each of the ships. Therefore, we will implement a Class Ship, which holds the ships as static flyweight objects. Its main responsibilities are to represents the objects ships, which therefore will have attributes like name, length, position etc. This brings us to two other classes that will be required. Ship Type and Position. The Positions will represent hold the starting and end position of a ship on the grid by having a row and column Index. The ShipType is an enumeration that holds all the information regarding each individual ship type as well as their quantity in the game.
   In the before created Class Position we just mentioned the grid. The grid is an essential Part of the game. It is representing the current state of the game. It represents where the ships are placed, where hits are registered in the grid all the location of the ships will be stored. The grid will also be a key Element of the Class Output, which handles the display of the grid after each turn. Each Element of the Grid will be of the Type Block. Block is therefore the next class that needs to be created in order to model the battleship game. The Class block depicts each single block in the Grid and contains the information if it is empty or if there is a ship placed. If the block is hidden, hit, or revealed.

   These named Classes should build the foundation for the game, depicting the board and providing the required objects and methods. However, the game needs to be played. Therefore, we will require a Class Player as well as a Class Computer. The Player Class and Computer Class are quite similarly with the difference that the "actions" of the Player are being handled through the input, whereas the computer does this randomly. They both have a fleet, Ocean grid and target Grid. Their Methods are held in the Class Competitor. The afore mentioned

Input is the last required Class. It handles, processes, and validates all the input and closely interacts with the Player and Position to do so.



| Ship | |
|---|---|
| Superclass(es): | |
| Subclasses: | |
| • holds static flyweight ship objects.<br>• represents the different types of ships<br>• holds length in an array, name and initial<br>• hold its positions<br>• manages own status by counting hits<br>• returns status | • ShipType<br>• Fleet<br>• Grid<br>• Player<br>• Computer<br>• Position<br>• Input |

| Player | |
|---|---|
| Superclass(es): | |
| Subclasses: | |
| • has a fleet and his oceangrid<br>• sets location for his ships on the oceangrid<br>• shoots to the targetgrid of the Computer | • Fleet<br>• Grid<br>• Computer<br>• Position |

| Fleet | |
|---|---|
| Superclass(es): | |
| Subclasses: | |
| • gets the ships of class Ship<br>• holds ships and sunk ships<br>• returns amount of ships of each list | • Player<br>• Computer<br>• Ship |

| Grid | |
|---|---|
| Superclass(es): | |
| Subclasses: | |
| • Saves the locations of the ships<br>• has access to all the blocks<br>• Checks if ship location is available | • Ship<br>• Position<br>• Block<br>• Player<br>• Computer<br>• Output |

| Block | |
|---|---|
| Superclass(es): | |
| Subclasses: | |
| • Depiction of a block in the grid<br>• holds type {Ship, Empty} and status {Hidden, Hit, Revealed}<br>• holds the reference to a ship if type is Ship<br>• returns type and status with booleans. | • Grid<br>• Ship<br>• Player<br>• Computer |

| Position | |
|---|---|
| Superclass(es): | |
| Subclasses: | |
| • represents a position in the grid<br>• holds column and row index<br>• has boolean equals, same column and same row methods. | • Grid<br>• Ship<br>• Input<br>• Player<br>• Computer |

| Input | |
|---|---|
| Superclass(es): | |
| Subclasses: | |
| • processes and validates the input of the human player<br>• translates Input into Grid coordinates<br>• checks wheather coordinates are equal to ship length | • Player<br>• Position |

| Game | |
|---|---|
| Superclass(es): | |
| Subclasses: | |
| • Checks if the game is still running<br>• alternates the draws between the human Player and the Computer<br>• checks for winning conditions | • Player<br>• Computer |

| Computer | |
|---|---|
| Superclass(es): | |
| Subclasses: | |
| • has a fleet and his oceangrid<br>• sets location for his ships on the oceangrid<br>• shoots to the targetgrid of the Player | • Fleet<br>• Grid<br>• Player<br>• Position |

| Competitor | |
|---|---|
| Superclass(es): | |
| Subclasses: | |
| • holds all methods of Player and Computer | • Player<br>• Computer |

| ShipType | |
|---|---|
| Superclass(es): | |
| Subclasses: | |
| • holds the information about each shiptype and its quantity | • Ship |

| Output | |
|---|---|
| Superclass(es): | |
| Subclasses: | |
| • output of the grid after each turn | • Grid |

2. **Following the Responsibility Driven Design, describe the main classes you designed to be your project in terms of responsibilities and collaborations.**

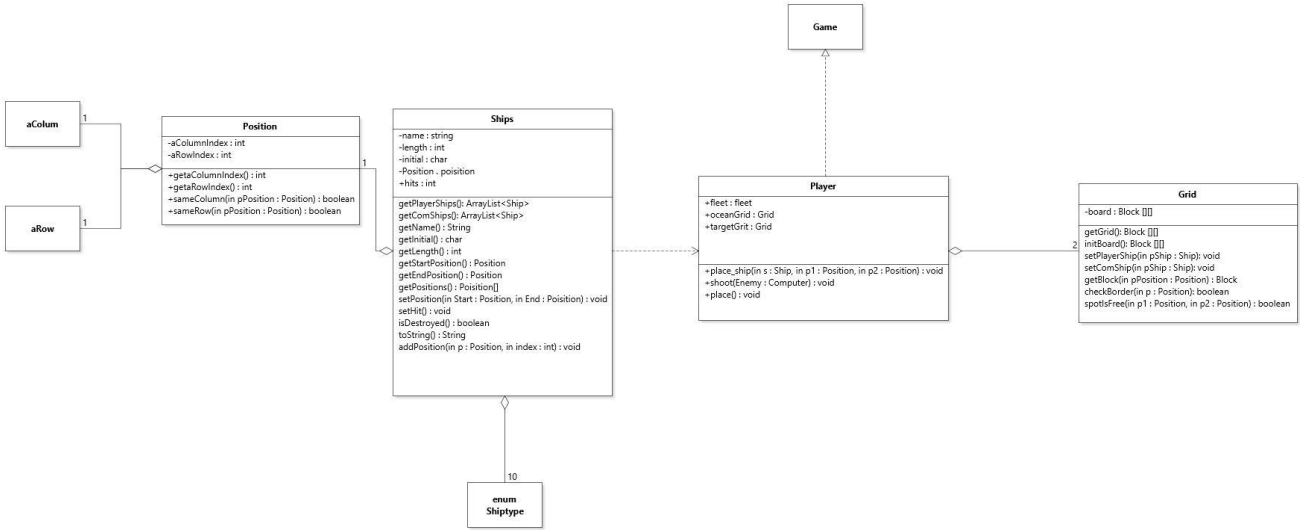
   The main classes will consist of the classes “Grid, Player, Ship and Position”, since they will be the most commonly used and therefore are the most fundamental building blocks for the battleship game. Due to that we expect them to have the highest interaction of methods between each other and determine them to be the main classes to be designed.

3. **Why do you consider the other classes as less important? Following the Responsibility Driven Design, reflect if some of those non-main classes have similar/little responsibility and could be changed, merged, or removed.**

Each Class has it´s importance and use, else they could be omitted. However, we identified the aforementioned Classes as key elements of the battleship game. The other Classes are required to provide Objects and methods for the main classes to function for the game to be played.

Regarding which non main classes could be modified we determine, that he Position Class might be merged with the Ship class, since the Position could be assigned as attributes to the ship Objects. Another thought might me to merge the classes of Player and Computer, since they they behave quite similarly. The merge could reduce the amount of code but make it less understandable. The separation of both however might make the implementation easier.

## 4. Draw the class diagram of the aforementioned main elements of your game



5. **Draw an object diagram to show the main elements of your game in a step of the game of your choosing**