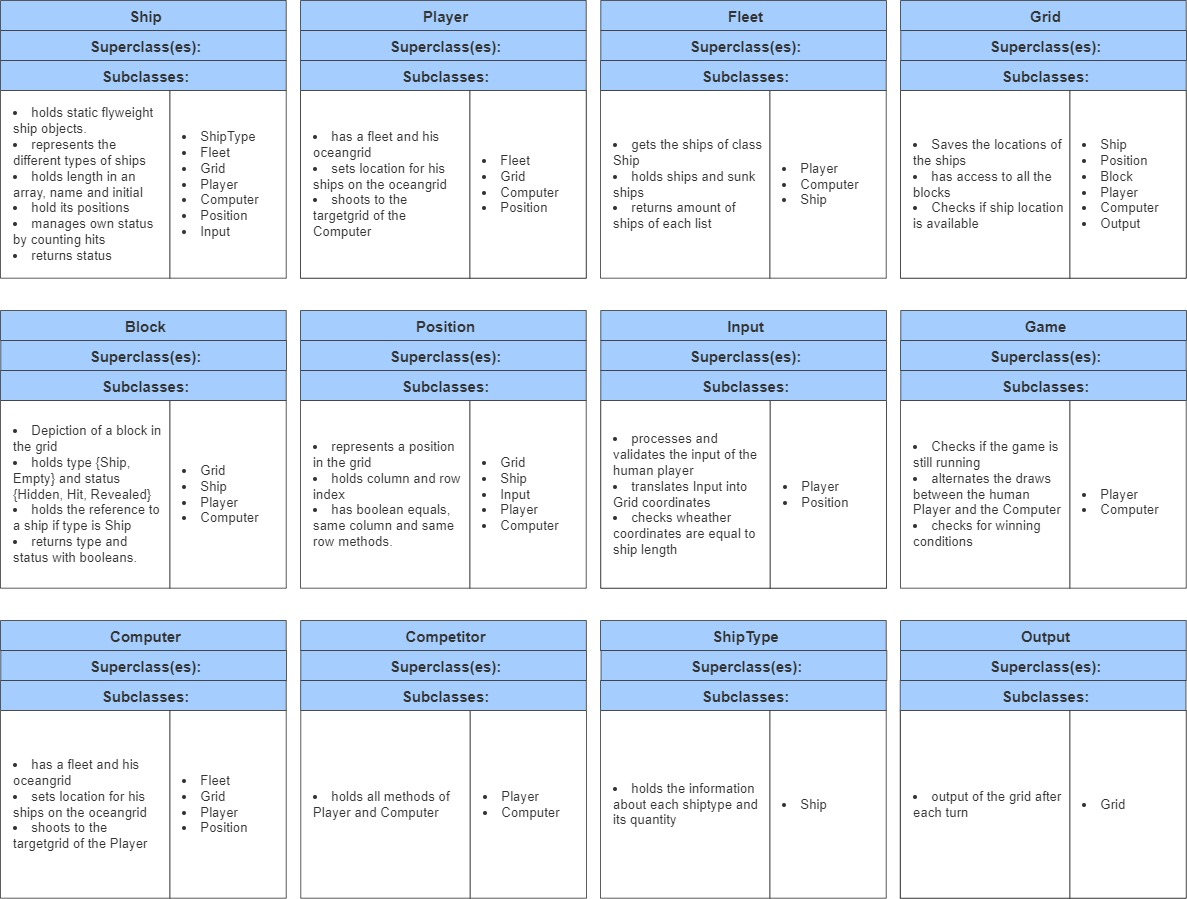
**Exercice 1**

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 Ships, 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 hold 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.



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

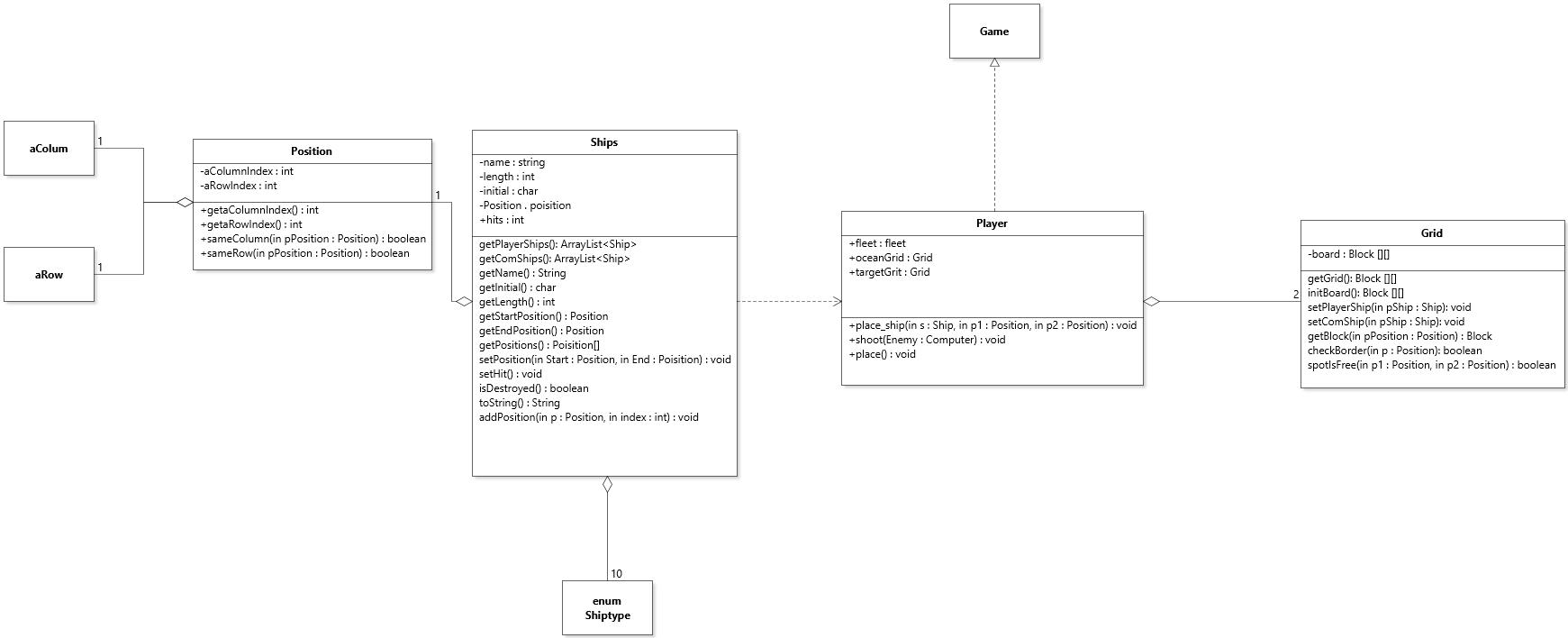
The main classes will consist of the classes “Grid, Player, Ship and Position”, since they will be the most commonly used and therefore are the most fundamental building blocks for the battleship game. Due to that we expect them to have the highest interaction of methods between each other and determine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

1. **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.

1. **Draw the class diagram of the aforementioned main elements of your game**



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

**Exercise 2**

The first step was to implement the main Classes.

The **class Ship** has a private constructor as well as private Attributes. The Attributes of the Class ship are its name, length, initial, positions and hits. All these attributes are private and except for the attribute hits final. There are various public methods to return the above-named attributes. The Player ships as well as the computer ships are stored in an array list as flyweight objects.

For the creation of the Objects, we require the **class ShipType**, which is an enum of all the ships and contains next to the names, initial and lengths of the ships additionally the exact number of ships of each type that a player requires to play the game.

The c**lass Position** has two attributes, which both are private and final. These Attributes are a column and a row Index. Methods to return these attributes have been implemented as well as methods to compare.

In accordance with the learned design patterns the equal method has been overwritten.

The next two classes that needed to be implemented were the Grid and Block classes. Objects of the class **Grid** are a two-dimensional Arraylist of blocks. The Class contains methods to place the Players ships as well as the computers ships. Other public methods are the return of an individual block, e check if a spot is free for the placement of a ship as well as check if a position is withing the boarders of the grid.

Objects of the class **Block** have private attributes. These attributes are type (either empty or a ship), status (hidden, revealed, or hit) a initial and a representation. The initial is received from the assigned ship which gets assigned after the placement on the grid. The methods in the Class Block consist of various public methods that for example Adjust the blocks attributes if a ship is assigned to it (either by the Player or the Computer), change the status to hit if a shot has been placed on a ship or miss if the block has no assigned ship. There are also multiple return methods to return attributes of the block as well as methods checking if a ship is assigned or if the spot on the grid is empty.

The next two classes are rather similar, and both implement the **interface Competitor**, which holds the methods for both classes. The classes are **Player** and **Computer**. Objects of these classes have three attributes. They have a fleet, a target grid and an ocean grid. All of them are private and finale. The constructor is private as well. Both have a private method place\_ship which places a ship on the respective Ocean grid. Additionally there are methods shoot, place and a method which checks, if the game is won (if the size of the fleet of a player is equal to zero). The method shoot, which is public as well handles the shooting of the player or the computer and modiefies the targeted block in the Grid accordingly.

The class **Fleet** is responsible to compare if all flyweight ship object have been sunken or not. Object of the class Fleet have two Attributes in form of an Arraylist containing Objects of the Class Ship. One Arraylist containing all the ships of a player, the other one containing the sunken ones. As son as the size of said Arraylists are equal, the entire fleet has been sunk. The methods in the Class either add sunken ships to the Arraylist of sunken ships or return the amount of sunken ships.

The **class Input** handles all the Input from the human player as well as the validation to make sure, that the input in within the for the game necessary and specified parameters. There are four methods in the Class Input. One of which checks if the length of the ship input by the “player” is correct. EnterShot handles the Shooting of the Player while inputShipPosition handles the placing of the Ships by the player on the Board.. The two remaining methods are ParsePosition and inputShipPosition. The remaining method ParsePosition is used inputShipPosition and EnterShot. ParsePosition validates the Input according to the games rules, and the requirements of the Design. If the input fulfils all required criteria a new Positionobject is returned.

The class **Output** is responsible for the graphical representation of the grid on the console. The two methods print and printGrid, of which the latter is private and my print called upon) create the output for the different Grids as well as the lines in-between. The separation of the two is by choice to be in line with the Encapsulation.

As last Class, the class **Game,** which alsocontains the main method needed to be implemented. The class Game Initiates the Game and alternates the moves between the human Player and the AI. Additionally it checks for winning conditions.

In Regards of the Design techniques learned in the lecture, multiple approaches were implemented. Next to the implementation of the ships as flyweight objects and the override of the equals and iterator methods input validation has been used multiple times. In general a approach of Design by contract approach has been chosen. The extensive commenting of methods should make the code more understandable and the intention of the Author clear. Many attributes and methods have been set to private and or final in attempt to achieve protection of mutation of the parameters by client code and to follow the Design pattern of encapsulation.