**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 array 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, the check if a spot is free for the placement of a ship as well as check if a position is within 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 of the interface Competitor, namely shoot, place and a method which checks, if the game is won (if the size of the fleet of a player is equal to the size of the sunk ships). The method shoot, which is public as well handles the shooting of the player or the computer and modifies 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 soon 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 is 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 by 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 Position object 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 also contains the main method needed to be implemented. The class Game initiates the Game and alternates the moves between the human Player and the Computer. 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 an 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.