# Description of implementation decisions

GameMaster & GameUtils:

As we were developing the classes, we realized that there are some global methods used by all classes that do not belong to a specific class per se. Those are convenience methods like

User Input:

We decided to use input validation within the functions which accept human user input (callShot and placeFleet) in favour of contract (for example do not allow invalid Coordinate to be created in the constructor). The reason is that we don’t know yet how to properly implement an error handling. So given the user inputs a wrong Coordinate (invalid, wrong format), the error would be thrown but not handled and the game ends.

With input validation we tried to catch all possible errors a user could commit to make sure the subsequent objects receive valid Coordinates. Subsequent objects rely on that validation and assume every Coordinate or list of Coordinates they receive was properly checked.

Player:

We decided against a Player interface in favor of an abstract class Player. Initially, we implemented an interface Player, but realized that the two implementations PlayerHuman and PlayerComputer only differ in the methods callShot and placeFleet, but have identical methods for receiveShot, getBoatTypeString, isHuman and hasLost.

So, an abstract class with both abstract and non-abstract methods seemed like a sensible way to go.