PROBLEM STATEMENT - BATTLESHIP[1]

Implement a console-based variation of the classical board game that you can play against the computer. To keep things simple, the game is played on a 6x6 grid as shown in the figure below. Before the game starts, both players place two battleships on the board, so that no part of the ships is outside the board and they do not overlap. Once the game starts, players take turns attacking squares, with hits tracked on the player and targeting boards. The game ends when one player hits all the squares occupied by the enemy ships. Program functionality is broken down as follows:

1. Place your battleships on the board using the following command: **`ship`** $<C_1L_1C_2L_2C_3L_3>$
   [E.g. commands **`ship C3D3E3`** and **`ship A0A1A2`** gives the ship position in the figure below]
   In case an invalid square is provided, a part of the ship falls outside the grid, or ships overlap (have at least one common square) the program will provide an error message and will not place the ship **[2p]**.

2. You can repeat the **`ship`** command as many times as you wish, until you are pleased with your ships' position on the grid. If you already placed two ships on the board, entering a valid command will replace the ship that was added first with the current one. **[1p]**

3. Each time the command results in valid placement of a ship on the player's board, the player board will be displayed as illustrated on the left hand side of the figure below. **[1p]**.

4. Start the game using the following command: **`start`**
   The start command can only be provided once two battleships have been placed on the player board. Once the game starts, the program will randomly place two battleships on the computer's board, using the same rules. **[2p]**

```
  A B C D E F        A B C D E F
0 + . . . . .      0 . . . . . .
1 + . . . . .      1 . . . . . .
2 + . . . . .      2 . . . . . .
3 . . + + + .      3 . . . . . .
4 . . . . . .      4 . . . . . .
5 . . . . . .      5 . . . . . .
 Player board       Targeting board
```

5. Play the game **[2p]**. The player and computer will attack squares in turn, with the player having the first attack. Attacks are made using the command: **`attack <square>`**. [E.g. **`attack E4`**]. When all the squares containing a player's ships are hit, the player loses the game. The program will provide a message in this regard.
   After each attack, the player and targeting boards are updated. Given the player and targeting boards above, the following series of attacks will result in the following possible board configuration:

```
attack E4                    A B C D E F          A B C D E F
Player misses!             0 + . . . . .      0 . . . . . .
computer attack C2         1 + . . . . .      1 . . . . . .
Computer misses!           2 + . o . . .      2 . . X . . .
attack C3                  3 . . + X + .      3 . . X . . .
Player hits!               4 . . . . . .      4 . . . . o .
computer attack D3         5 . . . . . .      5 . . . . . .
Computer hits!                Player board        Targeting board
attack C2
Player hits!
```

6. Cheats. We want to know where the computer's battleships are. Using the **`cheat`** command, the program will reveal the placement of the computer's ships (it is up to you to decide how) **[1p]**.

Non-functional requirements:

- Implement a layered architecture solution.
- Provide specification and tests for the methods involved in functionalities 1 and 2. In case no specifications or tests are provided, these functionalities are graded at 50% value.

[1] Subject in 2016/2017 retake exam ☺