PPA Assignment 9

This coursework is designed to test the content from Weeks 1-5 and 7-10.

Do you think there is a problem with any of the content below? Let us know immediately at programming@kcl.ac.uk.

Read through this brief carefully before starting your attempted solution and make sure you follow it as closely as possible, paying particular attention to the names required for elements of your code. Also ensure that you comment your code, as part of the assignment documentation.

You are advised to always access the latest version of this brief through KEATS.

For this week's assessment, consider the following scenario, and then complete the tasks that follow it:

In the game of Battleship, each player positions a set of ships at a set of unknown grid references, and then take it it turns to guess their opponent's grid references, in order to sink their opponent's ships (Reference: https://en.wikipedia.org/wiki/Battleship_(game)). Each ship has multiple parts, each of which is destroyed when a hit is specified at the same row and column as that part.

In this assignment, we will consider a simplified version of the game, which is shown in Figure 1, in which only one player (our program) positions the ships, and the other player (the player) guesses (purely numerical) grid references. Moreover, we will assume that ships are only positioned horizontally, from left to right, and always have their leftmost part at column zero.

1. Model this scenario based on the following requirements:

    1. First model a ship `Part`:

        1. As we can see from Figure 1, each part of a ship exists at a specific `row` and `column` number.

        2. As we can also see from Figure 1, a part can either be `destroyed` or hidden (i.e. not destroyed). It should be possible to `set` a part to be destroyed, and understand when a part `is` destroyed.

        3. Model that one `Part` can be compared to another `Part`, and these parts are considered equivalent when their `row` and `column` values match.

        4. A part should have a string representation that, based upon whether that part of the ship is destroyed or still hidden to the player, shows either a hit symbol ("[X]") or a blank symbol ("[ ]"), respectively. This is shown in Figure 2.

        5. To receive full marks for this question, there should not be any way to access the `row` or `column` values from outside the `Part` class.

2. Next, model a `Battleship`:

    1. As can be seen from Figure 1, a battleship consists of a number of `parts`.

    2. Initially a battleship does not consist of any `parts`. So, when a battleship is first created, these parts need to be created, and stored as `parts`, in order to represent the row and columns occupied by the ship. As can be seen from Figure 1, in our simplified game, all the parts of a ship are on the same `row`, which should be specified when the ship is first created. Given this row, N parts should be created on this row, in the appropriate columns, where N is the size of the ship. For example, in the case of the battleship shown in Figure 1, 5 `Part`s should be created and added as `parts`:

        1. Part 1: Row = 0, Column 0.
        2. Part 2: Row = 0, Column 1.
        3. Part 3: Row = 0, Column 2.
        4. Part 4: Row = 0, Column 3.
        5. Part 5: Row = 0, Column 4.

    3. A battleship can be `hit` at a particular `row` and `column`. If these values correspond to a `part` that exists within the ship's `parts`, then this part should be `set` as destroyed, and `hit` should confirm that part of the ship has been destroyed. If the part of the ship to which the `row` and `column` values pertain has already been destroyed, `hit` should simply confirm the previous hit. Otherwise, this method should confirm that the hit has failed.

    4. Model that one ship can be compared to another ship, and the two are the same when the number of `parts` in one ship is the same as the number of `parts` in another ship. Indeed, this fact can be seen from Figure 1. However, if all the parts in this other ship have been destroyed, then it is not considered equal to the current ship, as it has been sunk. Similarly, if a destroyed ship is compared to a floating ship, even if they are of the have the same number of parts, they should not be equal. Ergo, two sunken ships of the same size are equivalent.

    5. A ship should have a default string representation, which returns all of its parts side-by-side, which is a sufficient representation as we only consider horizontally positioned ships in our simplified game.

    6. To receive full marks for this question, there should not be any way to access the number of parts in a ship from outside the `Battleship` class.

3. Create a `Cruiser` which is a type of `Battleship` that is defined as having 4 parts.

4. Create a `Frigate` which is a type of `Battleship` that is defined as having 3 parts.

5. Create a `Minesweeper` which is a type of `Battleship` that is defined as having 2 parts.

    1. Model that there is a 50% chance that, when hit, the resilient minesweeper will not be damaged by an opponent's hit.

6. Next, create a `Board`:

    1. Given that our ships contain their own row and column information in the form of `parts`, there is no need for us to explicitly model a board as 2D plane. Instead, a board is simply defined by a list of `Battleship`s (`ships`).

    2. When a board is created, the ships shown in Figure 1 should be added to the board at the correct rows: 1 Battleship, 2 Cruisers, 1 Frigate and 1 Minesweeper. It should be possible to `get` the ships currently on the board.

3. It should be possible for a `hit` to be specified on a board, in the form of a `row` and `column` value. `hit` should check each ship to see whether that hit has been successful or not, and return a positive result if it has, and a negative result if it has not.

4. The board should have a default string representation, as shown in Figure 2, that shows each ship (with each part, which is either hidden, or hit), and any empty locations. To achieve this, consider the following:

    1. Give that we know that, in our simplified version of the game, each row of the board will only contain a single horizontally positioned ship, we can make each ship from `ships` appear on a new line in our string representation, in order to get an accurate representation of the board, as shown in Figure 1.

    2. For those locations on the board not occupied by part of a ship, a blank character ("[ ]") should be added. Again, because of the simplified nature of our game, to do this, it might be useful to understand the type of each ship that is positioned in each row (i.e. on each line), so that it is possible to simply enter a suitable number (5 - N) of blank ("[ ]") symbols after the string representation of each ship on each row.

2. Create a class `Game`, which can be compiled and run from the command line. Use this class to do the following (in order), using the classes and methods you have created for Question 1.

1. First of all, create the board.

2. At the start of the main game loop, print the number of ships of each type that are currently on the board (i.e. those ships that have not been sunk), as shown in Figure 2. To do this, you may wish to look at the static method *frequency* in the *Collections* class.

3. Then, print the board.

4. Next prompt the user to enter a single row and column number on a single line, separated by a space, again as shown in Figure 2.

5. Check whether that row and column number constitutes a hit on the board, if it does, print "Hit!", if it does not, print "Miss!".

6. The loop should continue until the user enters "quit".

7. *For simplicity, we will not consider the case in which the user enters anything but a single row and column number separated by a space, or how to proceed once the game has ended, but you are welcome to add this (unmarked) functionality.*

Once completed, submit your assignment using the link marked `Assignment 9: Nexus Submission Link' on KEATS.

You must complete the plagiarism and collusion training before submitting this assignment.

You must also submit complete documentation of your solution. You will find a sample piece of documentation in the Support section on KEATS marked `Sample Assignment Documentation'. Submit your documentation using the link marked `Assignment 9: Documentation Submission' on KEATS.

Students who do not submit documentation along with their code, or vice-versa, will receive a mark of zero.

Any submitted code or documentation that is found to be unduly similar to the code or documentation submitted by any other student(s), will result in a penalty for those involved.

Provisional mark for your code will be released on KEATS within one week of submission. Final assignment grades will be submitted to the exam board at the end of the semester, and will take into consideration the quality of your documentation and the quality of the comments written into your code directly.

For all other queries, see the Support section on KEATS, specifically the document marked `Introduction'.