Assignment 7: Classes and Interface

Assignment 7 (30 points) (**Due date/time**: Thursday, March 31st, 11:59 PM.)

You are to program a simplified version of the Battleship guessing game. The assignment is broken into two parts. The below game description applies to both parts.

Battleship game description. The field (ocean) is a square 7x7 grid. One of the coordinates of the grid is a number (from 1 to 7) and the other is a letter (from 'a' to 'g'). Your program should randomly place a fleet of seven ships in the ocean. Each ship takes up exactly one location in the ocean. Multiple ships CANNOT be placed in the same location. The ships, however, can be placed in adjacent locations. The user fires on the ships by specifying the coordinates of the shot. The program reports whether each shot was a hit or a miss. If the shot was a hit, the ship is sunk. The game continues until all ships are sunk. **Note:** The program does not keep track of the locations of the previously fired shots (will be part of future work).

In this assignment, most likely, you will need to implement 3 classes. Those classes have to implement interfaces described below (so make sure that you design the interfaces and then the classes ☺)

-Potential Classes you may create:

1: class Location: `// coordinates (location) of the ship and shots and implements the interface` **`LocationAPI`**`.`

`It has 4 potential methods:`

```
1: constructor location() // void constructor that assigns -1 to X
//coord, and * to Y coord
```

`The three other methods are declared in the following interface:`

`LocationAPI``: (`***`interface`***`), has the following signatures:`

```
2: void pick(); // picks a random location
3: void fire(); // asks the user to input coordinates of the next shot
4: void print(); // prints location in format "a1" (y and x)
```

`Additionally, class Location has the following 3 data members:`

```
final int fieldSize=7; // the field (ocean) is fieldSize X fieldSize
int x;  // 1 through fieldSize
char y; // 'a' through fieldSize
```

2: class **Ship**: implements the Interface **ShipAPI** and contains ship's coordinates (location) and whether it was sunk.

It has 4 potential methods:

```
1: ship() // void constructor, sets sunk=false
```

The other methods are declared in the following interface:

ShipAPI: ***(interface)***

```
boolean match(location) // returns true if this location matches
                           // the ship's location
boolean isSunk()// checks to see if the ship is sunk
void sink();       // sets "sunk" member variable of the ship to true
void setLocation(location) // deploys the ship at the specified
//location
void printShip() // prints location and status of the ship
```

Additionally, the class has 2 other data members:

```
location loc;
boolean sunk;
```

3: class Fleet: contains the fleet of the deployed ships, and implements the **FleetAPI** interface.

FleetAPI: ***(interface)*** contains the following signatures:

```
void deployFleet() // deploys the ships in random locations
                      // of the ocean
boolean operational() // predicate returns true if at least
                         // one ship in the fleet is not sunk
boolean isHitNSink(location) // returns true if there was a deployed
                      // ship at this location (hit) and sinks it
                      // otherwise returns false (miss)
void printFleet() // prints out locations of ships in fleet

int check(location) // returns index of the ship
                               // that matches location
```

Additionally, the class other 2 data members:

```
final int fleetSize=7 // number of battleships

ship ships[fleetSize]        // battleships of the fleet
```

Hint: To randomly assign a character coordinate (from 'a' to 'g') for the location in `pick()`, randomly select a number from 1 to 7 and then use a `switch` to select the appropriate letter. You also can add the character 'a' to a randomly generated number from 0 to 6 to get a random letter from 'a' to 'g'.

Example: The following code fragment generated 10 random letters from a to z:

```
Random rand = new Random();
char randomChar;
for (int i = 0; i < 10; ++i) {
      randomChar = (char) ('a' + rand.nextInt(27));
      System.out.println(randomChar);
}
```

Hint 2: The logic of `initialize()`, `check()` and `deploy()` is very similar to the logic of lottery number selection we studied in class.

- **Functions that display the location of the fleet.**

The printout is done using the following functions.

  - `printShip()` prints the location and status of a single ship
  - `printFleet()` prints the location and the status (sunk or not) of the whole fleet of ships. This function uses `printShip()`. The output of this function might look like:

```
b5 sunk, c3 up, a2 up, e1 sunk, g6 up
```

Projects, you need to create

1. **Test**. Create a project titled `Lab7_Test`. Define member functions for the classes given above (do not forget to implement interfaces) and place the defined member functions in proper java files.

You should implement the member functions for classes incrementally. Start with implementing member functions for class location, then proceed to ship and then to the fleet.

Try to design a simple test class to test your classes (`TestShips.java`). Once your project works correctly with all code of test, submit your first part (`Lab7_Test`).

2. **Game.** Create a project titled `Lab7_Game`. Use the classes use implemented and create the game program (`class Game`) that contains `main()`, invokes the game functions declared before and tested in a test project and implements the Battleship game.

Hint: the pseudocode for your `main()` should be as follows:

```
declare an object of the class fleet in the Game or test class.
call deployFleet() method on it
prompt the user if ships' positions and status need to be printed (testing)
loop while method operational() returns true (i.e. at least one ship is not
sunk.)
   Inside the loop,
   declare an object of class location,
   invoke fire() on it to get the location of the user's shot,
   pass this object to isHitNSink() method of your fleet object
   analyze the return value of this method and report a hit or a miss
   if requested printing ships' status
        invoke printFleet() // again, this is just for testing
```

Make sure your programs adhere to the proper programming style. Submit your project to the designated dropbox on Canvas.