Level 1- Position

The Position class encapsulates the simple concept of a coordinate with x and y values. The x and y values must be greater than 0. If a value for either x or y less than 0 is passed to the constructor then the constructor must throw an IllegalArgumentException.

Level 2 – ConsoleSunkenShipReporter

When a Ship sinks it will notify all of its observers (Classes implementing ShipObserver) that it has sunk. The Ship will pass itself as an argument to the sunk method of ShipObserver.

The ConsoleSunkenShipReporter simply reports information about the sinking to the console via a PrintStream.

Note on Registering ConsoleSunkenShipReporter with Ship

In order to be notified of when a Ship is sunk the ConsoleSunkenShipReporter must register a ShipObserver with the Ship. This is done through the watchShip(Ship ship) method. The pattern provided to you is of considerable help here.

In case you were wondering: This approach of using an anonymous inner class which implements ShipObserver removes the need to declare ConsoleSunkenShipReporter as implementing ShipObserver and prevents "publication" of the public methods in the ShipObserver interface.

```
private ShipObserver shipObserver = new ShipObserver() {
    @Override
    public void sunk(Ship ship) {
        // Do stuff
    }
};

public void watchShip(Ship ship) {
    ship.addShipObserver(shipObserver);
}
```

Level 3 – OpponentBoardTest

The OpponentBoard represents the board in Battleships that tracks hits or misses of mines fired in the opponent's grid. In the design the OpponentBoard observes an ObservableBoard via the BoardObserver interface. An OpponentBoard is registered with an ObservableBoard via the register(ObservableBoard observableBoard) method. This is the same pattern as the ConsoleSunkenShipReporter registering a ShipObserver with a Ship in level 2.

The OpponentBoard must hold a 2D CoordinateState array that is assumed to be dimensioned PlayerBoard.BOARD_WIDTH * PlayerBoard.BOARD_HEIGHT (hint: OpponentBoard extends AbstractBoard and AbstractBoard may well be a good place to store this 2D array…). When a hit or miss is to be reported the observed Board will call the appropriate BoardObserver hit or miss method passing the relevant Position. These in turn must update the OpponentBoard's 2D CoordinateState array.

Level 4 – ShipTest

Among the provided fields in the Ship class are a Heading and Position. These fields are key to describing the ship. The Heading is one of North (towards max y coord), South (towards min y coordinate), East (towards min x coordinate) or West (towards max x coordinate).

Along with the four fields provided by default additional fields may be useful…

The Ship must be able to hold references to zero or more ShipObservers which must be notified when the ship has sunk. It must also be possible to remove ShipObservers.

Level 5 – PlayerBoardTest

The PlayerBoard represents the board where you place your ships at the beginning of a game and track the position and hit/miss state of the mines fired by your opponent.

The PlayerBoard must allow for adding and removing of BoardObservers.

A crucial method in PlayerBoard is the addShip(Ship ship) method. Here validation must be conducted to ensure that the Ship will not hang off the “edge” of the board. Reminder, the board indexing is 0 based (X: 0 to PlayerBoard.BOARD_WIDTH – 1 and y: 0 to PlayerBoard.BOARD_HEIGHT – 1). It also must be ensured that the Ship being added will not overlap any Ships that have already been added to PlayerBoard.

Just like OpponentBoard, PlayerBoard keeps track of a 2D CoordinateState grid which is again assumed to be dimensioned PlayerBoard.BOARD_WIDTH * PlayerBaord.BOARD_HEIGHT.

Level 6 – ConsoleBoardDrawerTest

The ConsoleBoardDrawer's job is to print an ASCII art representation of a DrawableBoard to a PrintStream. You've made it this far. You're on your own now…