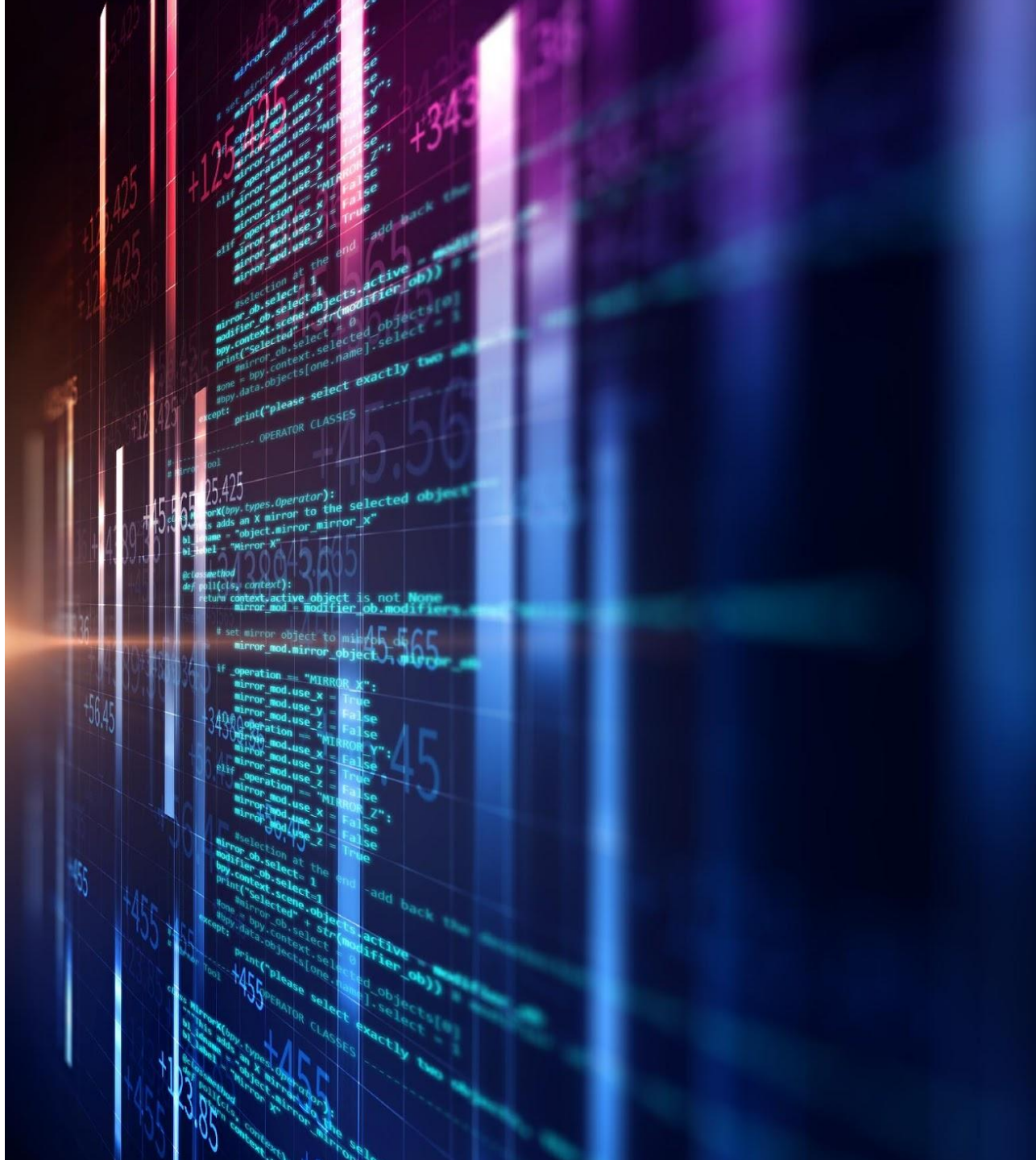
About Battleship

Brandon Krakowsky

Penn Engineering



About Battleship

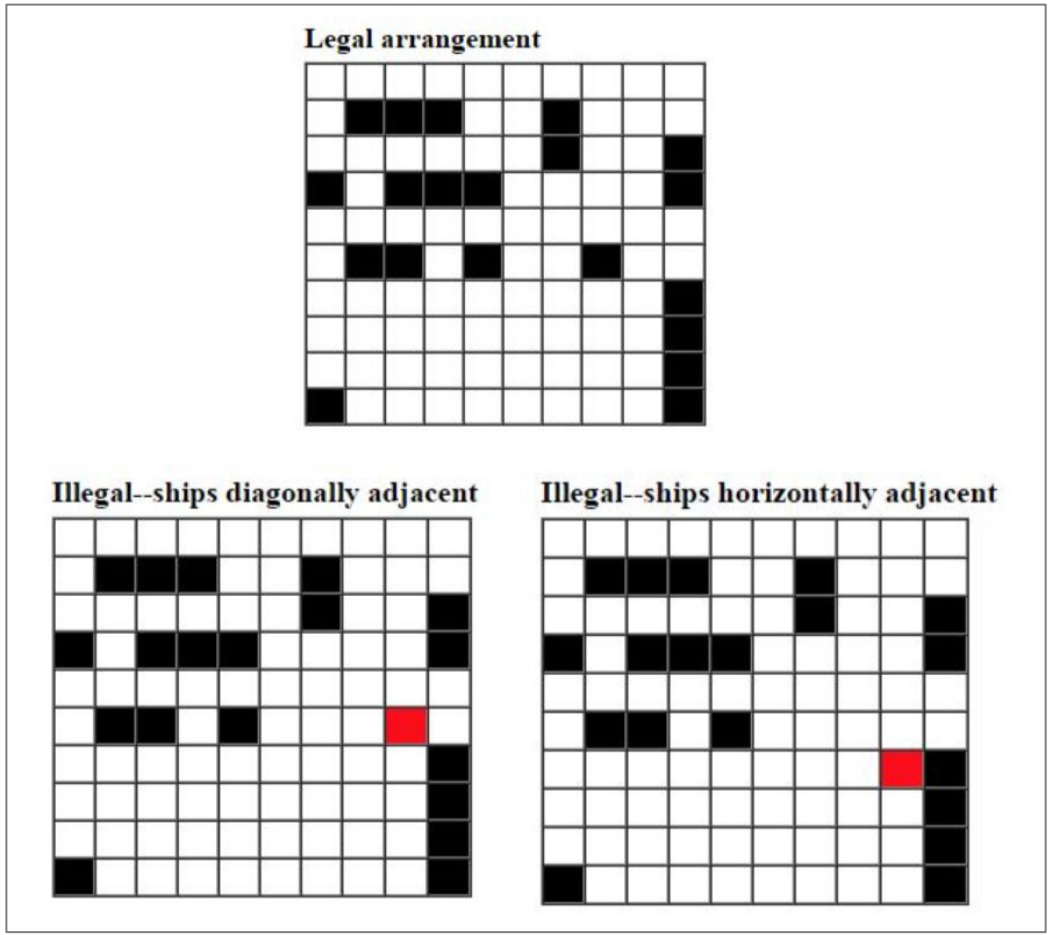
The Ocean & Placing Ships

- The game is played on a 10 by 10 “ocean”
  - This is represented by a 2-dimensional array
  - It’s defined in the Ocean class

```
private Ship[][] ships = new Ship[10][10];
```

- The Ocean class randomly places different kind of ships in the array (the “ocean”)
  - It does this with the *placeAllShipsRandomly* method
  - Ships should be placed in a location such that they don’t touch each other

The Ocean & Placing Ships



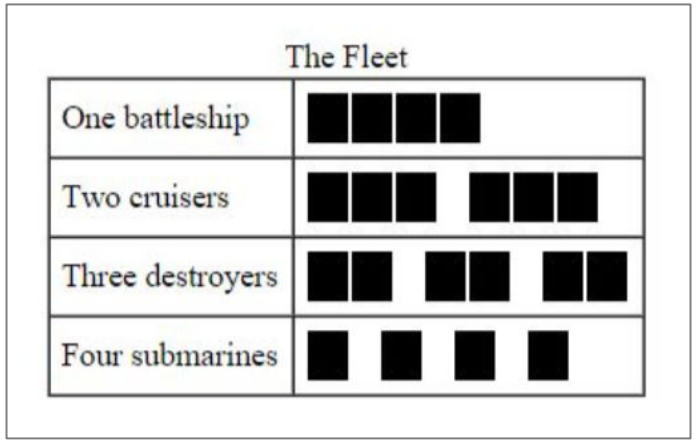
The Ocean & Placing Ships

- You can check if a ship can be placed in a particular location using the *okToPlaceShipAt* method in the Ship class
  - This method is inherited by other ship types extending the Ship class
- You place a ship in a particular location using the *placeShipAt* method in the Ship class
  - This method is also inherited by other ship types extending the Ship class

Placing Ships Based on Length & Direction

- Different types of ships have different lengths
  - For example, Cruisers have a length of 3
  - When you place a Cruiser in the 10 by 10 array (the “ocean”), it will take up 3 adjacent slots
- If it’s placed horizontally, it will take up 3 slots next to each other, *in a single row* of the array
  - Here’s the logic

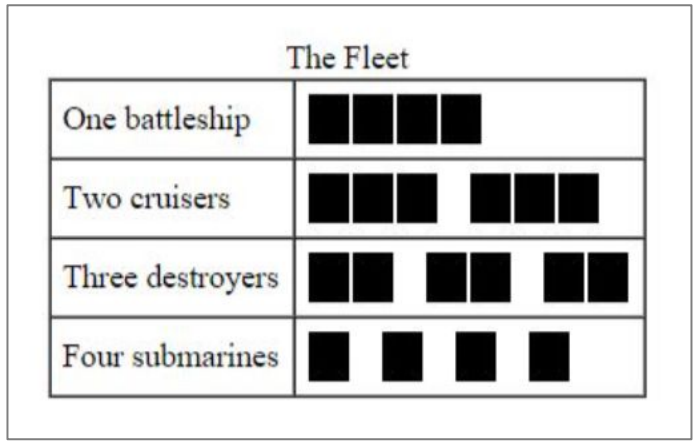
```
Cruiser cruiser = new Cruiser();
ships[2][1] = cruiser; //reference to cruiser
ships[2][2] = cruiser; //reference to cruiser
ships[2][3] = cruiser; //reference to cruiser
```



Placing Ships Based on Length & Direction

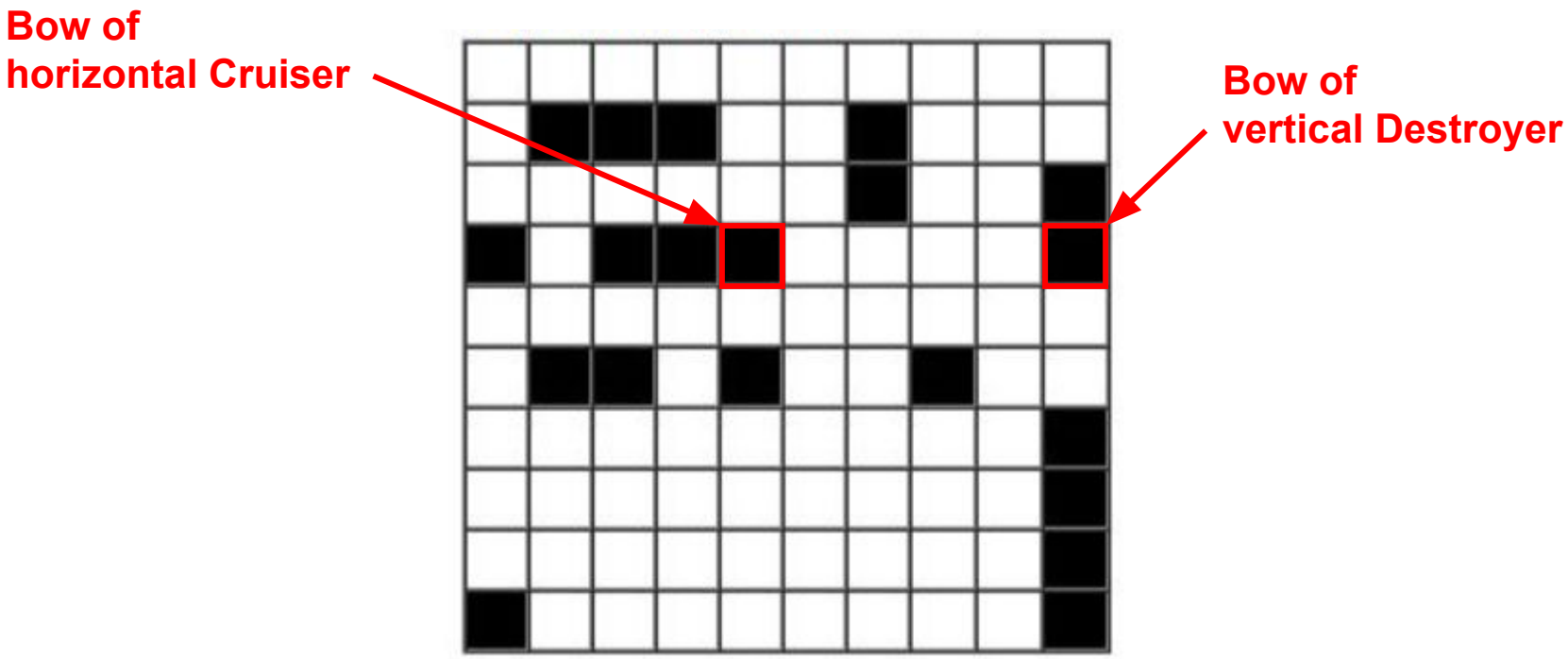
- If it's placed vertically, it will take up 3 slots next to each other, *in a single column* of the array
  - Here's the logic

```
Cruiser cruiser = new Cruiser();
ships[2][1] = cruiser; //reference to cruiser
ships[3][1] = cruiser; //reference to cruiser
ships[4][1] = cruiser; //reference to cruiser
```



The Bow of a Particular Ship

- For placement consistency, let's agree that horizontal ships face East (bow at right end) and vertical ships face South (bow at bottom end).



Firing At & Hitting Ships

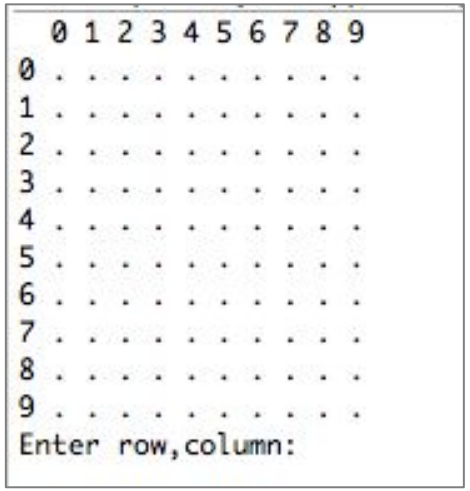
- The BattleshipGame class controls the game
  - It gets user input of 2 int values (comma separated), to indicate a location in the 10 by 10 array (the “ocean”) to shoot at
  - It fires at that location using the *shootAt* method in the Ocean class
  - It prints the current state of the ocean
  - It does this until the game is over (when all ships have been sunk) within a *while* loop

Firing At & Hitting Ships

- The *shootAt* method in the Ocean class will call the *shootAt* method in the ship (or EmptySea), in that particular location
  - If it hits the ship, a boolean value of true will be placed in the boolean[] hit array, in that particular location
  - For example:
    - If you hit the front part (bow) of a cruiser, then hit[0] = true
    - If you hit the back part of a cruiser, then hit[2] = true
- Once all of a ship's locations have been hit, the ship has been sunk

Printing the Ocean

- The 10 by 10 array (“ocean”) is printed after each turn
- It uses the *print* method in the Ocean class and the *toString* method in the Ship class
  - The *toString* method will be inherited by every other type of ship
  - The EmptySea class will override *toString*
- Print the ocean as a 10 by 10 array, with numbers at the top indicating the columns, and numbers on the left indicating the rows



Printing the Ocean

- Slots which have never been fired upon are displayed as a “.”
  - This can be taken care of in the Ocean *print* method
- Slots which have been fired upon, but where there is no ship, are displayed as a “–”
  - This can be taken care of in the EmptySea *toString* method
- Slots which have been fired upon, and have hit a ship, are displayed as an “x”
  - This can be taken care of in the Ship *toString* method
- Slots which have been fired upon, and have hit and sunk a ship, are displayed as an “s”
  - This can also be taken care of in the Ship *toString* method

Printing the Ocean

- Here's the logic

  for each location in the 10 by 10 array (the "ocean")

  - if the location contains a ship that is sunk
    or if the location has been shot at, and was hit or nothing was found
    - print the ship itself -- this will call *toString* in the Ship class or any ship subclass which has *toString* defined (i.e. EmptySea)
  - otherwise print "."

Printing the Ocean

- Here's a display of the ocean after 2 shots have missed, and 1 has hit a (real) ship.

```
  0 1 2 3 4 5 6 7 8 9
0 . . . . . . . . . .
1 . . . . . . . . . .
2 . . . . . . x . . .
3 . . . - - . . . . .
4 . . . . . . . . . .
5 . . . . . . . . . .
6 . . . . . . . . . .
7 . . . . . . . . . .
8 . . . . . . . . . .
9 . . . . . . . . . .
Enter row,column:
```