



Liam Hollins, Jacob Felknor, Atul Dhungel

CSCI 4448

Abstract

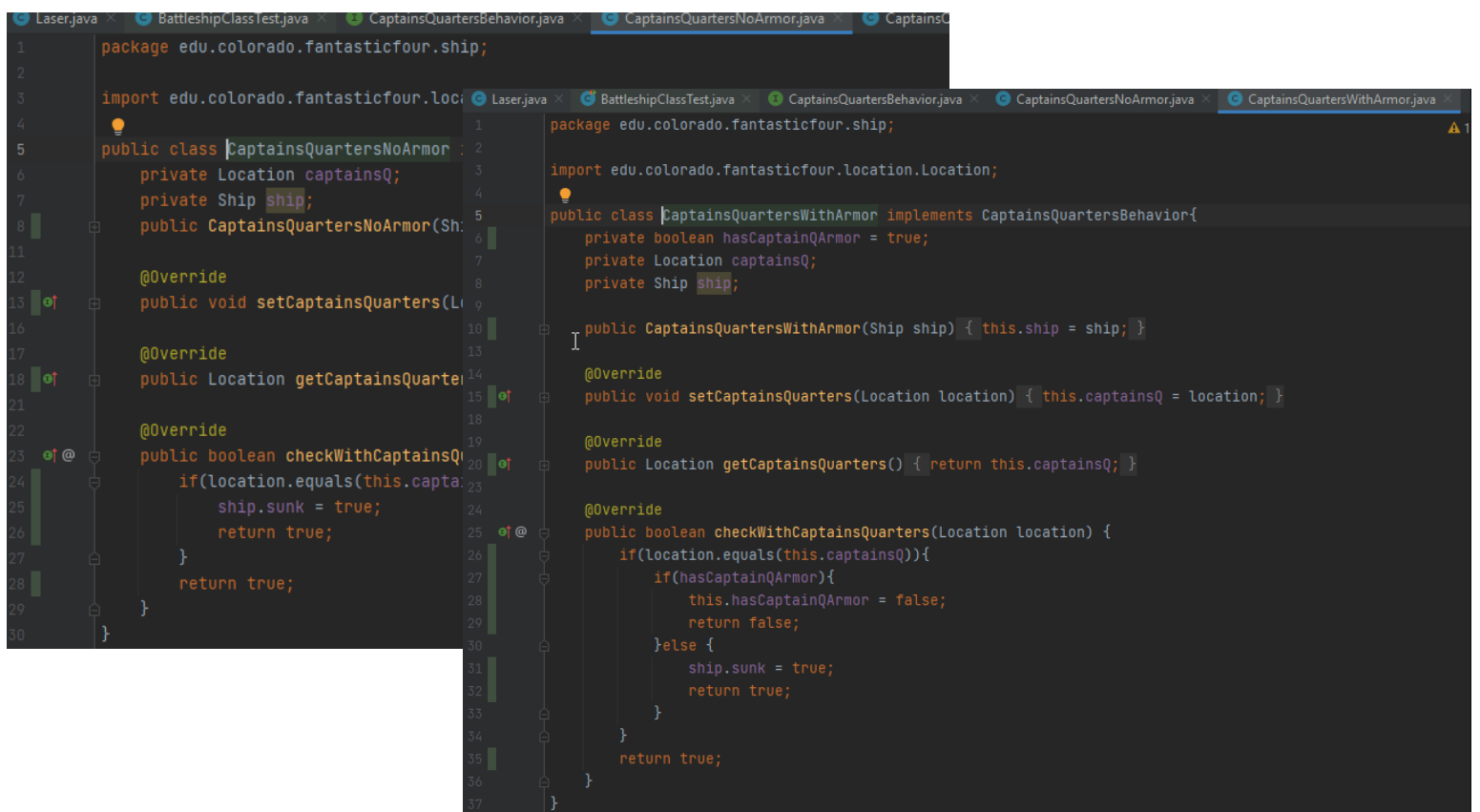
Our project is an implementation of the classic Battleship board game. Using object oriented programming in Java, we developed a computer terminal based version that incorporates all the rules from the original game, in addition to some new features.

Throughout the process of development, as we added new features and built up the playability of the game, we used design patterns refactoring techniques to help make our code readable and extensible. Additionally, through the use of Test Driven Development, we were able to achieve high quality program design and flexibility.

Development

In the first iteration of the project we had to implement a basic layout of our plan, which included designing our CRC cards, creating our github repository and becoming familiar with IntelliJ. We also created a team document to improve accountability. Early on, the team struggled with IntelliJ. This consisted of technical issues such as having proper path files, environment issues, etc.

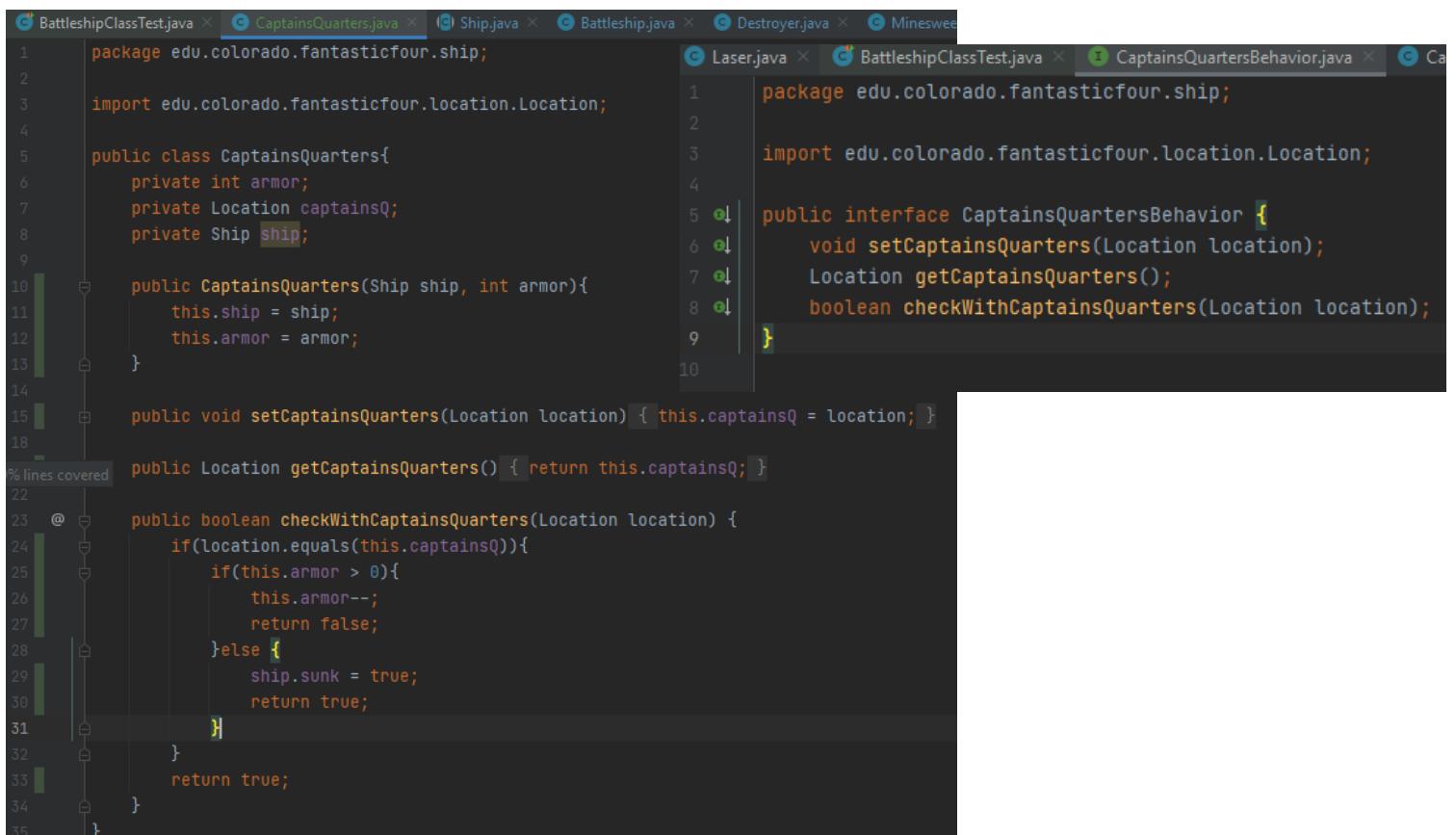
During the second iteration, we used pair programming and test driven development to set up the code skeleton for our base classes and prove that our logic worked. At this stage, one of our teammates unfortunately dropped the class and we had to rethink our team strategies. Junit was a relatively new tool for most of us, so test driven development was challenging. Jacob & Atul was assigned the task of working on Game, Board, and Cell class. Jason and Liam were assigned the task of creating Player and Ship class.



```
1 package edu.colorado.fantasticfour.ship;
2
3 import edu.colorado.fantasticfour.location.Location;
4
5 public class CaptainsQuartersNoArmor {
6     private Location captainsQ;
7     private Ship ship;
8     public CaptainsQuartersNoArmor(Ship ship) { this.ship = ship; }
9
10    @Override
11    public void setCaptainsQuarters(Location location) { this.captainsQ = location; }
12
13    @Override
14    public Location getCaptainsQuarters() { return this.captainsQ; }
15
16    @Override
17    public boolean checkWithCaptainsQuarters(Location location) {
18        if(location.equals(this.captainsQ)) {
19            ship.sunk = true;
20            return true;
21        }
22        return true;
23    }
24 }
25
26 package edu.colorado.fantasticfour.ship;
27 import edu.colorado.fantasticfour.location.Location;
28
29 public class CaptainsQuartersWithArmor implements CaptainsQuartersBehavior {
30     private boolean hasCaptainQArmor = true;
31     private Location captainsQ;
32     private Ship ship;
33
34     public CaptainsQuartersWithArmor(Ship ship) { this.ship = ship; }
35
36     @Override
37     public void setCaptainsQuarters(Location location) { this.captainsQ = location; }
38
39     @Override
40     public Location getCaptainsQuarters() { return this.captainsQ; }
41
42     @Override
43     public boolean checkWithCaptainsQuarters(Location location) {
44         if(location.equals(this.captainsQ)) {
45             if(hasCaptainQArmor) {
46                 this.hasCaptainQArmor = false;
47                 return false;
48             } else {
49                 ship.sunk = true;
50                 return true;
51             }
52         }
53         return true;
54     }
55 }
```

At milestone 3 we ran into issues with how we were going to represent the spaces on the game board. We implemented a Location class that we used to abstract to different coordinates systems. We also Switched the captains quarters implementation to use delegation with a CaptainsQuartersBehavior interface by implementing classes called CaptainsQuartersWithArmor and CaptainsQuartersNoArmor.

Around milestone 4, our documentation and wiki began to become more consistent and it made each iteration smoother as we all knew which areas needed more focus. We continued to add features and use design patterns to develop well tested and readable code. We also began looking for code smells and incorporating refactoring techniques we learned in lecture. One such refactor was the Captain's Quarters, which we modified to maintain a variable armor value rather than a representing the armor with two other classes.



```
1 package edu.colorado.fantasticfour.ship;
2
3 import edu.colorado.fantasticfour.location.Location;
4
5 public class CaptainsQuarters{
6     private int armor;
7     private Location captainsQ;
8     private Ship ship;
9
10    public CaptainsQuarters(Ship ship, int armor){
11        this.ship = ship;
12        this.armor = armor;
13    }
14
15    public void setCaptainsQuarters(Location location) { this.captainsQ = location; }
16
17    public Location getCaptainsQuarters() { return this.captainsQ; }
18
19    public boolean checkWithCaptainsQuarters(Location location) {
20        if(location.equals(this.captainsQ)){
21            if(this.armor > 0){
22                this.armor--;
23                return false;
24            }else {
25                ship.sunk = true;
26                return true;
27            }
28        }
29        return true;
30    }
31 }
32
33
34
35
```

```
1 package edu.colorado.fantasticfour.ship;
2
3 import edu.colorado.fantasticfour.location.Location;
4
5 public interface CaptainsQuartersBehavior {
6     void setCaptainsQuarters(Location location);
7     Location getCaptainsQuarters();
8     boolean checkWithCaptainsQuarters(Location location);
9 }
10
```

Requirements and specifications

1. 10x10x2 3-dimensional virtual game board
2. Three surface combatant war ships
 - a. Minesweeper
 - i. 2 units long
 - ii. Can be sunk with one hit to Captains Quarters
 - b. Destroyer
 - i. 3 units long
 - c. Battleship
 - i. 4 units long
3. One sub-surface ship
 - a. Submarine
 - i. 4 units long with an additional unit on side
 - ii. Exists a level below the other ships
4. A series of weapons available to Players
 - a. Bomb
 - i. Default weapon
 - ii. 1 damage to ships on surface
 - b. Laser
 - i. Enabled after player sinks one of opponent's ships
 - ii. 1 damage to ships on surface or subsurface
 - c. Sonar
 - i. Enabled after player sinks one of opponent's ships
 - ii. Allows players to view a detailed map of a specified area on their opponent's play area.
 - d. Minefield
 - i. Placed at the beginning of the game
 - ii. Deal damage to ships if players move into them
 - e. Doubleshot
 - i. Replaces Bomb as default weapon when player only has one ship remaining
 - ii. Allows for two shot per turn
5. Ability to move/unmove fleet
6. Menu based interface
7. Network Play

USER STORIES/USE CASES

MILESTONE 4:

1. Laser

-> once player has sunk an opponent's ship, the laser will be updated to the player.attackWeapon. Then on the next turn, the laser class will call useAt function and the shot will check for any ships on the surface or below. Results are updated using the notifyObservers(), note that this will sink any submarines at the given location with a z coordinate < surface

2. User takes shot with default weapon

-> player class calls takeShot(), which sets attack weapon and calls useAt function on a given location. useAt checks if the location is below surface, if not call notifyObservers function to check if the shot hit/miss/sunk. If hit, the shipGPS class updates the result of the hit.

MILESTONE 5:

1. Minefield

-> weapon type starts as 'mine'. User enters coordinates where they wish to place mines on opp's board. Mines coupled to opp's board, when opp places ships, they receive damage if placed on mine. Opp's ships also receive damage if fleet is moved into mines.

2. Doubleshot

-> with one ship remaining, user is given double shot weapon type to use if they choose. When used, user provides two coordinates to take shots

Architecture and Design

- The system is largely based around the Player class, which contains the fields and methods which help describe the state of the player's progress throughout the game.
- The Game class operates the basic turn function and deals with the system IO. Two players will belong to a game object for each time the game is played.
- Board class consists of checking whether a ship is on board. It uses composition of Cell class.
- Cell class gets the result of a shot that the player decided to target. It has a ship class as a composition.
- Weapon class has a subclass consisting of Bomb, Laser, Sonar and Minefield. Minefields are placed on the board at the beginning of the game when the players are placing ships. The Bomb is a default weapon that we give to the Player. The player can use it to take shots at the enemy. The bomb cannot hit submarines. Laser is a weapon that gets unlocked after sinking one ship. The laser can hit all ships. Sonar can be used twice and gets unlocked after the player sinks a ship.
- Ship class has ShipGPS class and CaptainQuarters class as composition. It has a subclass of Battleship, Destroyer, Minesweeper, and Submarines. Each of these subclasses got different sizes. Submarine has to be placed underwater so it's coordinate placement consists of (x,y,z). ShipGPS class tracks ships and CaptainsQuarters deals with placement of ships. If CaptainQuarters gets hit, the ship sinks.
- Observer design pattern is used to update the status of the ships. If the ships have been hit or missed then it notifies us.
- Location class deals with coordinates of ships. Location class allows users to get x,y,x coordinates.
- To summarize, Board has a cell. Cell has a ship. Ship has a GPS. GPS has an observer. Different types of weapons have different capabilities.

Personal Reflection

Liam:

This project was great practice for me in developing my object oriented programming skills. Learning Java was a bit of a challenge at the beginning, as was using TDD to write our code, but I think once I became more comfortable with these tools I was able to visualize how adding design patterns to a project like this made the code more flexible and fluid. It was also difficult to lose Jason early on, but I think we had a great group that was able to step up and fill in the role of a fourth person. By actively communicating, and being proactive for deadlines we were successful in the collaboration aspect of this assignment. Overall this was an enjoyable process and I think I've learned a lot, not just about OOAD, but about working as a team on a software project. I think the experience I've gained working on the Battleship game will help me a lot in my future career.

Jacob:

I thoroughly enjoyed working on this project with this team. We all communicated very well throughout the semester and our meetings were productive. I've known simple Java for about a year now, but it was really a lot of fun to be able to work on a larger project from the ground up including testing and applying design patterns in addition to sharpening my skills. I also liked learning how to use a full fledged IDE like IntelliJ as well. Simple things like changing a function name were much easier using its features.

This project gave me valuable experience for how professional software should be developed and I'll take the lessons with me as I start my career.

Atul:

Overall, I thought that this class helped me become a better programmer. The scope and depth at how I think when I am implementing a design has increased for the better. Before taking this class, I would code without thinking about the design at all. I always took into consideration runtime complexities of functions but never really took into account the entire code itself. I have also become more disciplined when it comes to making test cases. We are always told that having test cases is important, and ofcourse it is, but this class made me **experience** the importance of it. Especially when our code got larger and larger, it became increasingly important to test after every change. The long term benefit of it is fruitful. I had never been exposed to UML or CRC cards but I definitely see why it's important to learn those, especially when you are designing large projects. To summarize, this class is what you make of it. If you put in large amounts of work and really try, you are going to reap lots of benefits. I enjoyed the class and thought I was exposed to some important topics and concepts that will help give me better odds at getting a job.