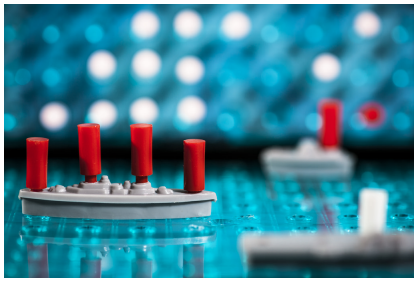
CSCI-605 Advanced Object-Oriented Programming Concepts

Homework 7 - BattleShip



1 Introduction

For this homework you will implement a simplified text-based version of the well-known game Battleship. Besides having no graphics, the game is simplified because it has only one player.

1.1 Goals

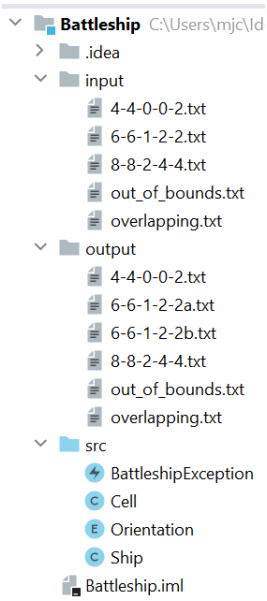
Students will gain experience working with:

- Input/Output Streams (java.io)
  - Text Streams
  - Object Streams
- Exceptions
  - Defining
  - Raising (throw)
  - Handling (catch)

Warning: The use of the `Scanner` class is not allowed for this homework!

2 Starter Code

Your project's structure should look as follows:



Here is a brief explanation of what is provided to you:

1. The `src` folder contains the following java files:
   - `BattleshipException`: A custom checked exception for representing different violations of the game's rules during runtime.
   - `Orientation`: A enum representing the different orientations (horizontal or vertical) of a ship on the board.
   - `Cell`: A class representing a cell on the board.
   - `Ship`: A class representing a single ship.

   These classes are incomplete and do not include all files that are required to be implemented (check the javadoc, sequence and class diagrams).
2. The **doc** folder contains the javadoc documentation for the `Battleship` class.
3. The **input** folder contains configuration files with different ships placements.
4. The **output** folder contains sample runs. One run was split into two parts where the game state was saved in the middle.

3 Implementation

The game will receive input from two different sources: a configuration file with the ships placement, and then, from the stardard input to play the game.

3.1 Configuration File

The ships placement on the board will be mandated in a configuration file. The content of the file is separated by spaces. The first line of the file will contain two numbers: first the height (number of rows) of the board, and next the width (number of columns).

Each additional line describes a ship placement, in four parts:

1. the number of the uppermost row on which the ship sits.
2. the number of the leftmost column on which the ship sits.
3. whether the ship is HORIZONTAL (spans columns) or VERTICAL (spans rows).
4. the length of the ship.

For example, look at the input file `4-4-0-0-2.txt`. This file describes a 4x4 board with two ships. Assume the file is formatted correctly with integer positive values.

```
4 4
0 0 HORIZONTAL 2
2 3 VERTICAL 2
```

3.2 Command line arguments

The program is run on the command line as:

```
$ java Battleship setup-file
```

If the number of arguments is incorrect, display a usage message and exit the program, i.e.:

```
Usage: java Battleship setup-file
```

If the number of arguments is one, then you have one more thing to do: check to see if the file is a binary file of an interrupted game (saved by using an `ObjectOutputStream`) or if it is a text file containing an initial configuration. Try to open the file first as an `ObjectInputStream`. If that fails (exception is thrown), try to open it as some kind of `Reader`. If that fails, print an error message and terminate the program.

If the number of arguments is correct, it is guaranteed the file will be formatted correctly.

3.3 User Commands

Once the file is set up, the game begins. The program will provide the following commands for playing the game:

- `help` - Display a list of all commands available.
- `h row column` - Hit a cell.
- `s file` - Save game state to file. (Serialization process)
- `!` - Reveal all ship locations.
- `q` - Quit game.

3.4 Error Handling

The errors you have to deal for this program are:

1. If the command line argument is not present, display a usage error and exit.
2. If the file does not exist or is not readable, display an error message and exit.
3. If the user attempts to hit the same cell more than once, display an error message. The program must allow the user to enter a new command.
4. If the user attempts to hit a cell outside the bounds of the board, display an error message. The program must allow the user to enter a new command.

5. Attempting to place a ship outside the bounds of the board, display an error message and exit.
6. Attempting to place a ship where there is already another ship, display an error message and exit.

3.5 Design

The game consists of a top-level `Battleship` class. It has access to a game board (`Board` class) instance, which, importantly, contains or refers to all data related to the current state of the game at any point in time. This capability is realized by the `Board` having access to all of the ships and all of the individual cells on its grid. For design convenience every cell is an object as well, and each cell on which a ship is placed has a reference to that ship.

You will also implement your own hierarchy of checked exceptions to represent different violations of the game's rules (see documentation). The provided `BattleshipException` java file will be at the root of your exception hierarchy.

We strongly suggest that you follow the supplied design for this homework. These UML diagrams detail the classes and their relationships. Here is how a single play of the game proceeds. (Not all details shown.)

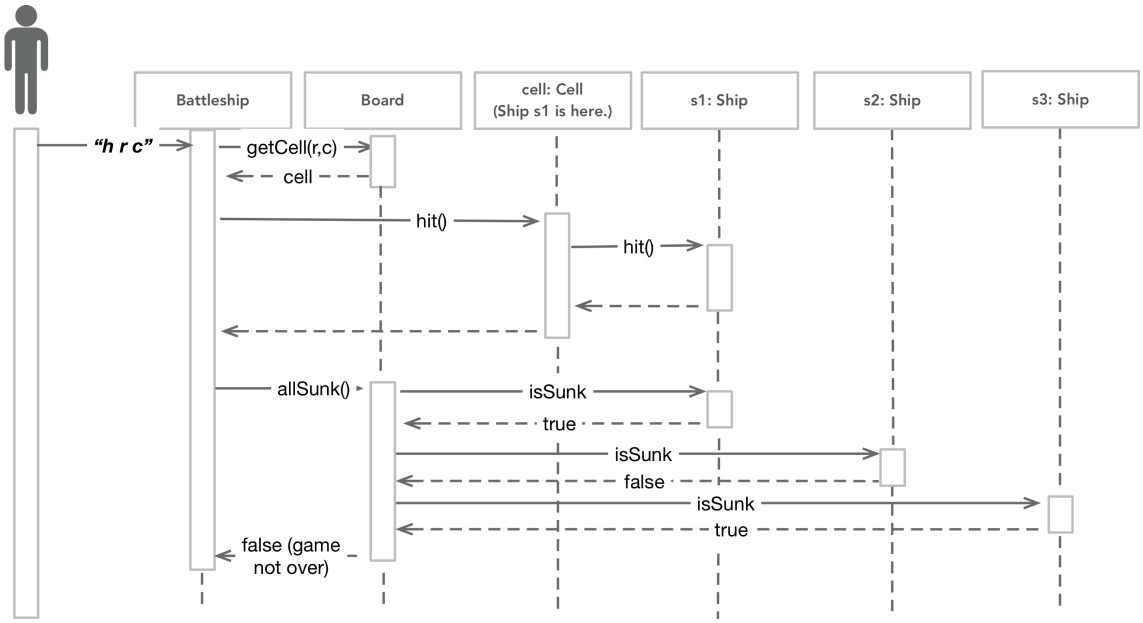


Figure 1: BattleShip Design Sequence Diagram

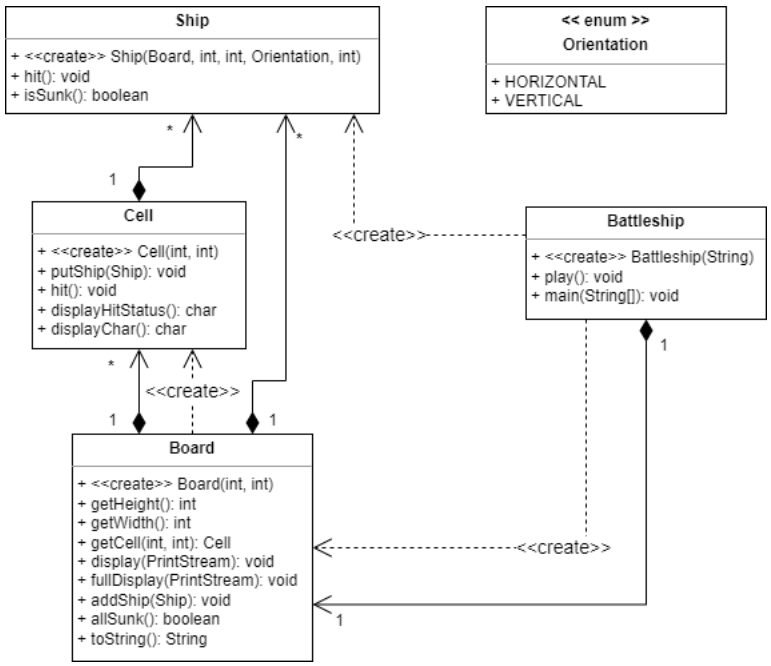


Figure 2: BattleShip Design Class Diagram

4 Submission

You will need to submit all of your code to the MyCourses assignment before the due date. You must submit your `src` folder as a ZIP archive named "hw6.zip" (if you submit another format, such as 7-Zip, WinRAR, or Tar, you will not receive credit for this homework).

5 Grading

The following deductions will apply:

- Completeness/Implementation: up to 100% if a part of the writeup is not observed.
- Correctness/Testing: up to 50% or affected part if the solution is not correct.
- Quality/Style/Documentation: up to 50% if solution is poorly designed.
- Explanation: up to 100% if the student is not able to respond grader's questions.