Lab 4: Battleships

Due: 18:00, 01 Mar 2012 (Fri) Full marks: 100

Introduction

In this lab, we are going to model a variation of the game called "Battleships"[1] using Object-Oriented Programming. The game requires two players. One player is called the "Commander" and the other player is called the "Captain". The Commander secretly placed a fleet of five ships on a 10 × 10 grid of squares, and the Captain tries to guess the location of the ships. Ships are denoted by rectangles of varying sizes. The following is the fleet of five ships and their respective sizes.

1. Aircraft Carrier – 5 squares
2. Battleship – 4 squares
3. Destroyer – 3 squares
4. Submarine – 3 squares
5. Patrol Boat – 2 squares

Ships can be place in any horizontal or vertical position (but not diagonally), and must not overlap with each other. Refer to the left of **Figure 1** for an example.

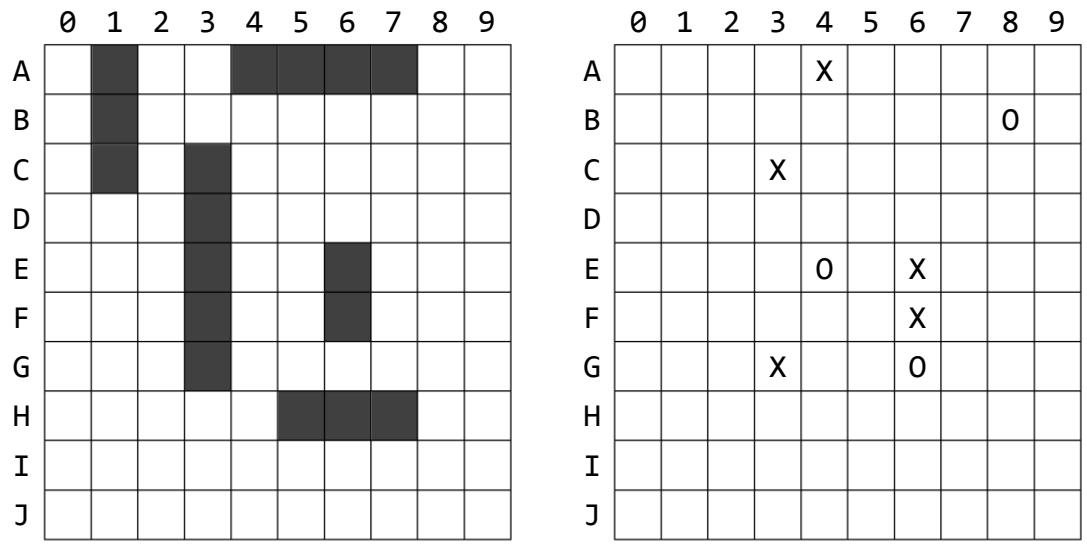

Figure 1: A Sample Battleships Game

The Captain tries to guess the location of all the ships by firing shots into the grid. For each shot, the Captain calls out the coordinates of a square (e.g. E4). The Commander responds with "hit" if a shot hits a ship and marked the corresponding

[1] URL: http://www.papg.com/show?1TMC

square with 'X'. Otherwise, it is a "miss" and marked the corresponding squares with '0'. Refer to the right of **Figure 1** for an example. A ship is sunk if the Captain has hit the last remaining squares of that ship, and the Commander must announce the name of the ship sunk. The Captain wins the game if all the ships of the fleet sunk within thirty shots; otherwise, the game is lost.

Program Specification

Create a new project in NetBeans named (`Battleships`), you have to implement two classes: **`Ship`** (`Ship.java`) and **`Battleships`** (`Battleships.java`) to model the game. Details of each class are given below.

`Class Ship`

This class models the ships in the game and should contain the following contents. You are *not allowed to add any extra* public *contents to the class*.

Private Instance Variables

- **`private char sType;`**
  The character representing the type of ship, namely:
  1. 'A' for Aircraft Carrier
  2. 'B' for Battleship
  3. 'D' for Destroyer
  4. 'S' for Submarine
  5. 'P' for Patrol Boat
- **`private char sPos, sRow, sCol;`**
  The characters representing the position ('H' for horizontal and 'V' for vertical) of the ship; and also the row ('A' to 'J') and column ('0' to '9') coordinates for the first square (i.e. leftmost for horizontal position; and topmost for vertical position) of the ship. For example, the Aircraft Carrier (5 squares) in **Figure 1** will take the values: **`sPos`** = 'V', **`sRow`** = 'C', and **`sCol`** = '3'.
- **`private int sDamage;`**
  The integer representing the damage (i.e. number of squares being hit) of the ship, and the value ranges from 0 to the size of the ship.

Public Constructor and Public Instance Methods

- **`public Ship(char type)`**
  The parameter **`type`** denotes the type of the ship. You should use it to initialize

the instance variable **`sType`**. This method should also initialize the instance variables **`sDamage`** to 0, **`sPos`** to 'V', **`sRow`** to 'A', and **`sCol`** to '0'.

- **`public String getName()`**
  This method returns the name of the ship according to the value stored in the instance variable **`sType`**:
  1. 'A' returns 'Aircraft Carrier'
  2. 'B' returns 'Battleship'
  3. 'D' returns 'Destroyer'
  4. 'S' returns 'Submarines'
  5. 'P' returns 'Patrol Boat'
- **`public void setLocation(char row, char col, char pos)`**
  The parameters **`row`**, **`col`**, and **`pos`** denote the characters for the row, column coordinates, and the position of the ship respectively. You should use them to set the value for the corresponding instance variables **`sRow`**, **`sCol`**, and **`sPos`**.
- **`public boolean isInBoundary()`**
  This method returns true if the ship is within the boundary of the 10 × 10 grid; returns false otherwise.
- **`public boolean isOverlap(Ship neighbour)`**
  This method returns true if the ship overlaps with another ship passing in as parameter **`neighbour`**; returns false otherwise.
- **`public boolean getShot(char row, char col)`**
  This method determines if a shot hits the ship or not. If the ship is hit, this method should increment the instance variable **`sDamage`** by 1 and returns true; returns false otherwise.
- **`public boolean isSunk()`**
  This method returns true if the damage of the ship is equal to its size; returns false otherwise.

Class Battleships

This class models the Battleships game and is also the main class. It is a client of class **`Ship`** and should contain the following contents. Again, you are *not allowed to add any extra* public *contents to the class*.

Private Instance Variables

- **`private char hitSymbol, missSymbol, emptySymbol;`**
  The character representing 'hit', 'miss' and 'empty' symbols for marking the grid during the game.

- **`private char[][] board;`**
  A two dimensional array of **`char`** representing the grid of squares and records the shots fired by the Captain during the game; the first dimension correspond to the row coordinate ('A' to 'J'), and the second dimension correspond to the column coordinate ('0' to '9'). For example, the element **`board[3][8]`** denotes the square D8 and the element **`board[2][5]`** denotes the square C5.
- **`private Ship[] ships;`**
  A one dimensional array of reference to **`Ship`** objects; used to represent the fleet of ships.

Public Constructor and Public Instance Method

- **`public Battleship(char hit, char miss, char empty)`**
  The parameters **`hit`**, **`miss`**, and **`empty`** denote the hit, miss, and empty symbols for marking the grid of square during the game. You should use them to initialize the instance variables **`hitSymbol`**, **`missSymbol`**, and **`emptySymbol`**. This method should also initialize the instance variable **`board`** as a 10 × 10 array of **`char`**.
- **`public void prepareForBattle(char[] fleet)`**
  This method implements part of the game flow where the Commander secretly placed a fleet of ships. The parameter **`fleet`** is a one dimensional array of **`char`** encoding the list of ships according to the type (see definition of ship type under Class **`Ship`**). Detailed steps for this method are given below.
  1. This method should first initialize the instance variable **`ships`** as a one dimensional array of **`Ship`** objects where the length is the same as the array **`fleet`**. Each element in the array s**`hips`** must reference a **`Ship`** object where the ship type is indicated by the respective element in the array **`fleet`**.
  2. This method should **repeatedly** display a dialog prompting the Commander to input the location for each ship of the fleet one-by-one, until all the ships has been placed. The input location must be encoded as a string of three characters representing the first square (i.e. leftmost for horizontal position; and topmost for vertical position) of the ship. The first character denotes the row, the second character denotes the column, and the third character denotes the position. For example, for the Aircraft Carrier (5 squares) in **Figure 1**, the Commander should input the string 'C3V'. For this input, you need to check if the Commander input is valid or not. An input is valid if *all* the followings are satisfied:
     a. The length of the input string must be three.
     b. The first character must between 'A' to 'J', the second character must

between '0' to '9', and the third character must either 'H' or 'V'.

c. The location where the ship is placed must not exceed the 10 × 10 grid. For example, you can put a Patrol Boat (2 squares) using the location 'I1V'. However, you cannot put an Aircraft Carrier (5 squares) on the same location as it would exceed the boundary of the grid.

d. No two ships of the fleet can overlap with each other.

If the Commander enters an invalid input, you need to display a warning message and ask the Commander to enter again until a valid input is made.

3. After the Commander has placed all the ships on a valid location, you should display a summary of the locations of all the ships.

- **`public void play(int numOfShots)`**

This method implements the flow for the Captain tries to guess the location of ships by making shots into the grid. The parameter **`numOfShots`** indicates the total number of shots the Captain can make in the game. Detailed steps for this method are given below.

1. This method should first initialize all the elements in **`board`** with the character **`emptySymbol`**.
2. Next, this method should **repeatedly** display a dialog prompting the Captain to input the shots one-by-one, until all the total number of shots fired reached **`numOfShots`**. The input shot must be encoded as a string of two characters encoding the row and column coordinates of the square being target, e.g. 'E5'. For this input, you need to check if the Captain input is valid or not. An input is valid if *all* the followings are satisfied:
   a. The length of the input string must be two.
   b. The first character must between 'A' to 'J', and the second character must between '0' to '9'.
   c. Each squares of the grid can only be shot once.

   If the Captain enters an invalid input, you need to display a warning message and ask the Captain to enter again until a valid input is made.
3. After a valid input has been input by the Captain, the program should determine whether the input coordinates hit a ship or not. If the shot hits one of the ship, you should mark the corresponding element in **`board`** with the character **`hitSymbol`**; mark the element with **`missSymbol`** otherwise. If the Captain hits the last remaining squares of a ship, that ship sunk and the program should display the name of the ship being sunk.
4. If the Captain has sunk all the ships before using up all the shots, a winning message should be displayed; a losing message should be display otherwise.

- **`public static void main(String[] args)`**
  The *static* main method is given to you and should be exactly like this.

```
char[] fleetOfShips = {'A', 'B', 'D', 'S', 'P' };
Battleships game = new Battleships('X', 'O', '.');
game.prepareForBattle(fleetOfShips);
game.play(30);
```

Programming Tips

The standard Java classes: **`String`**, **`StringBuilder`**, and **`JOptionPane`** are useful. Find in the API specification (http://docs.oracle.com/javase/7/docs/api/) to see which methods are related and useful. In all the dialogs, you can assume that *the user does not click the "Cancel" or "Close window" buttons*.

The default font for **`JOptionPane`** is variable width. You can use the following example code as a guide on how to change the font of **`JOptionPane`** into fixed width, so that you can display the grid in a nice format.

```
StringBuilder sb = new StringBuilder();
sb.append("1 2 3 4 5").append("\n").append("A B C D E");
JTextArea textarea = new JTextArea( sb.toString() );
textarea.setFont(new Font("Monospaced", Font.PLAIN, 14));
String input = JOptionPane.showInputDialog( textarea );
```

Sample Program Run

Refer to the provided sample runs in CU e-Learning System.

Submission

Submit the two source files `Ship.java` and `Battleships.java` to CU e-Learning System (i.e. the entry "**lab04 – Battleships**" under "**Lab Works**").

== END ==

