Assignment 9

Samuel Godwin, Computer Science BSc, K1630575

December 11, 2016

1 Introduction

The purpose of this assignment is to create a simplified game of Battleship in which my driver class 'Game' first creates an object of my 'Board' class (which acts as the board on which my game of Battleship can be played). Subsequently, my driver class contains a loop that controls the main flow of my game of Battleship. There are four types of ship ('Battleship', 'Cruiser', 'Frigate' and 'Minesweeper'). We will assume that ships are only positioned horizontally, from left to right, and always have their leftmost part at column zero. In my simplified version of the game, my program positions the ships at unknown grid references and the other player guesses numerical grid references to try and sink parts of the ships – each ship has multiple parts, each of which is destroyed when a hit is specified at the correct reference and, when all parts to a ship are destroyed: so is the ship.

(Pseudocode on p2).

2 Pseudocode

Part Class

```
CLASS 'Part':
    Create int field 'row'
    Create int field 'column'
    Create boolean field 'isDestroyed'
    METHOD 'Part'<row, column>:
        Set field 'row' to <row>
        Set field 'column to <column>
    END METHOD.
    METHOD 'setIsDestroyed<isDestroyed>':
        Set field 'isDestroyed' to <isDestroyed>
    END METHOD.
    METHOD 'getIsDestroyed':
        Return isDestroyed
    END METHOD.
    METHOD 'equals'<Object part>:
        IF part not instance of Part THEN:
            Return false
        ELSE:
            Return row == ((Part)part).row AND column == ((Part)part).column
        ENDIF.
    END METHOD.
    METHOD 'toString':
        IF isDestroyed == false THEN:
            Return "[ ]"
        ELSE:
            Return "[X]"
        ENDIF.
    END METHOD.
END CLASS.
```

Battleship Class

```
Import 'ArrayList'

CLASS 'Battleship':
    Create boolean field 'isDestroyed'
    Create ArrayList<Part> 'parts'
    METHOD 'Battleship' <row, N>:
        Set field 'isDestroyed' = false
        Initialise ArrayList 'parts' as new ArrayList<Part>
        FOR all index i in 'parts'
            Add new Part(row, i) to 'parts'
    END METHOD.
    METHOD 'Battleship'<row>:
        This(row, 5)
    END METHOD.
    METHOD 'hit'<row, column>:
        Create int variable 'result', set = 0
        Create new Part(row, column) 'partSupplied'
        Create boolean variable 'partMatches', set = false
        FOR EACH Part in 'parts'
            IF partSupplied == part:
                Set partMatches = true
                IF 'isDestroyed' in Part == false:
                    Set 'isDestroyed' in Part = true
                    result = 1
                ELSE:
                    result = 2
                ENDIF.
        ENDFOR.
        IF partMatches == false:
            result = 3
        ENDIF.
```

```
        Go to 'isDestroyed'
        Return result
END METHOD.
METHOD 'equals'<battleship>:
        IF <battleship> is an instance of 'Battleship' THEN:
                Return false
        ELSE:
                IF both are destroyed OR both are not destroyed THEN:
                        Return size of 'parts' == size of 'parts' in <battleship>
                ELSE:
                        Return false
                ENDIF.
        ENDIF.
END METHOD.
METHOD 'toString':
        Create String variable 'output' = ""
        FOR all index n in 'parts':
                output = output + parts.get(n)
        ENDFOR.
        Create int variable 'toAdd' = 5 – (size of 'parts')
        FOR int i = 0; index i < toAdd; i++:
                output = output + "[ ], "
        ENDFOR.
        Return output
END METHOD.
METHOD 'isDestroyed':
        Create int variable 'destroyedParts' = 0
        FOR EACH Part in 'parts':
                IF Part is destroyed THEN:
                        destroyedParts += 1
                ENDIF.
```

```
            ENDFOR.
            IF destroyedParts == size of ‘parts’ THEN:
                Go to ‘setIsDestroyed’, set <isDestroyed> = true
                Return true
            ELSE:
                Return false
            ENDIF.
        END METHOD.

        METHOD ‘setIsDestroyed’<isDestroyed>:
            Set field ‘isDestroyed’ = <isDestroyed>
        END METHOD.

        METHOD ‘getIsDestroyed’:
            Return isDestroyed
        END METHOD.
END CLASS.
```

Cruiser Class

```
CLASS 'Cruiser' extends 'Battleship':
    METHOD 'Cruiser' <row>:
        super(row, 4)
    END METHOD.
```

Frigate Class

```
CLASS 'Frigate' extends 'Battleship':
    METHOD 'Frigate' <row>:
        super(row, 3)
    END METHOD.
```

Minesweeper Class

```
Import 'Random'

CLASS 'Minesweeper' extends 'Battleship':
    METHOD 'Minesweeper' <row>:
        super(row, 2)
    END METHOD.
    METHOD 'hit'<row, column>:
        Create new Random 'rnd'
        Create int variable 'result', set = 0
        IF next pseudorandom int through 'rnd' == 1 THEN:
            super.hit(row, column)
        ENDIF.
        ELSE IF super.hit(row,column) == 1 THEN:
            Print "The shot was on target, BUT the resilient minesweeper did not receive
            damage on this turn!"
        ELSE IF super.hit(row,column) == 2 THEN:
            Print "The shot was on target, BUT that part of the ship has already been
            destroyed and the minesweeper takes no damage on this turn!"
        ELSE:
            result = 3
        ENDIF.
        Return result
    END METHOD.
END CLASS.
```

Board Class

Import ‘ArrayList’

Import ‘Collections’

CLASS ‘Board’:

Create ArrayList<Battleship> ‘ships’

METHOD ‘Board’:

Initialise ArrayList ‘ships’ as new ArrayList<Battleship>

Add new Battleship(0) to ‘ships’

Add new Cruiser(1) to ‘ships’

Add new Cruiser(2) to ‘ships’

Add new Frigate(3) to ‘ships’

Add new Minesweeper(4) to ‘ships’

END METHOD.

METHOD ‘getShips’<index>:

Return ships.get(index)

END METHOD.

METHOD ‘hit’<row, column>:

Create new Part(row, column) ‘partSupplied’

Create boolean variable ‘partMatches’, set = false

FOR EACH Battleship in ‘ships’:

IF Battleship.hit(row, column) != 3 THEN:

Set partMatches == true

Go to ‘hit’<row, column> in Battleship

ENDIF.

ENDFOR.

IF partMatches == true THEN:

Return true

ELSE:

Return false

ENDIF.

END METHOD.

METHOD ‘toString’:

```
        Create String variable 'output' = ""
        FOR all index n in 'parts':
            output = output + parts.get(n)
        ENDFOR.
        Return output
    END METHOD.
    METHOD 'getFrequencies':
        Create new Battleship object 'battleship', set <row> = 0
        Create new Cruiser object 'cruiser1', set <row> = 1
        Create new Cruiser object 'cruiser2', set <row> = 2
        Create new Frigate object 'frigate', set <row> = 3
        Create new Minesweeper object 'minesweeper', set <row> = 4
        Create int 'freqBattleship' = Collections.frequency of 'battleship' in 'ships'
        Createt int 'freqCruiser' = Collections.frequency of 'cruiser1' in 'ships'
        Create int 'freqFrigate' = Collections.frequency of 'frigate' in 'ships'
        Create int 'freqMinesweeper' = Collections.frequency of 'minesweeper' in 'ships'
        Print freqBattleship
        Print freqCruiser
        Print freqFrigate
        Print freqMinesweeper
END CLASS.
```

Game Class

Import ‘Scanner’

CLASS ‘Game’:

METHOD ‘main’:

Create instance of ‘Scanner’ called ‘in’

Create object of ‘Board called ‘board’

Create String variable ‘userInput’ = “”

DO…

Go to ‘getFrequencies’ in ‘board’

Print result of ‘toString’ in ‘board’

Print instructions

Scan for user input, store in userInput

Create int variable ‘row’ = -1

Create int variable ‘column’ = -1

IF userInput is “exit” THEN:

Break

ENDIF.

Set row = parseInt of userInput

Scan for user input, store in column

IF result of ‘hit’ in ‘board’ == true THEN:

Print “Hit!”

ELSE:

Print “Miss!”

ENDIF

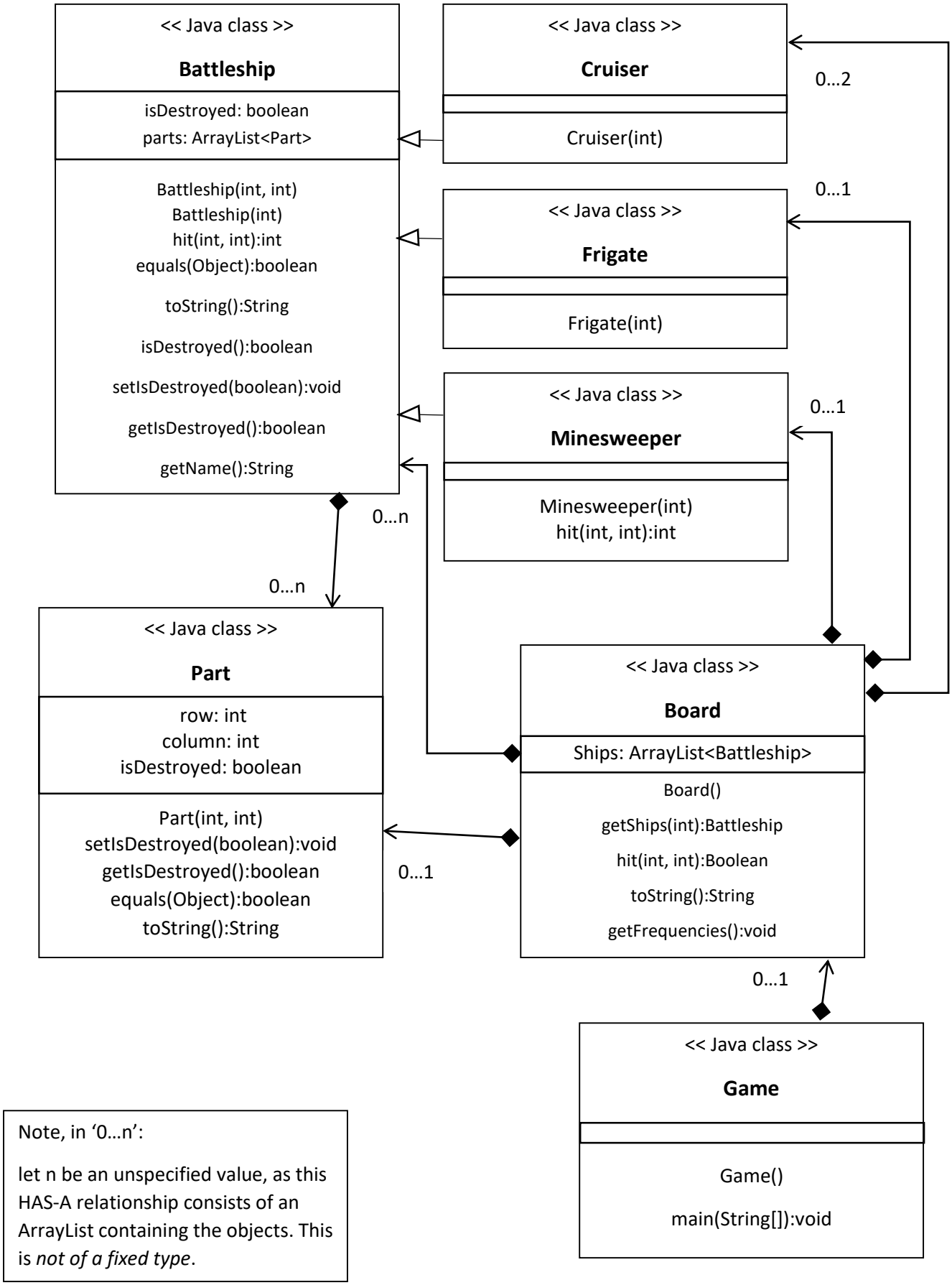
…WHILE userInput is not “exit”

Call the ‘close’ method on my ‘Scanner’ instance

END METHOD.

END CLASS.

3 Class Diagram



Note, in '0…n':

let n be an unspecified value, as this HAS-A relationship consists of an ArrayList containing the objects. This is *not of a fixed type*.

4 Description

My solution to this assignment involves first creating a ‘Scanner’ object in my driver class ‘Game’. I also create an object of my ‘Board’ class, ‘board’. After then initialising a variable of type ‘String’ = “”, I start a do-while loop. Within this loop, I initially call the ‘getFrequencies’ method through my board object. In my ‘Board’ class, my ‘getFrequencies’ method first creates a new variable containing an explicit, separate object for each of my ships. I next use the static method ‘frequency’ from ‘Collection’ in the Java class library, checking for the frequencies of each of these different ships, which uses my ‘equals’ method (made within my ‘Battleship’ class) which determines whether ships are equal based on their size and whether or not they are destroyed. I store these various frequencies in separate integer variables for each different type of ship. This determines the number of each type of ship that are on my board, after the construction of my ‘ships’ ArrayList in the constructor method within ‘Board’. After this, I print each of the frequencies (the number of each type of ship still in play) to the user by printing the aforementioned integer variables. Back in ‘main’, I print the contents of board via my overridden ‘toString’ method and then proceed to take input from the user as well as initialise two integer variables (‘row’ and ‘column’) each to a useless number, -1. These will be changed shortly.

At this point in my do-while loop, I use an if statement to check that the user did not enter “exit”. If they did, I ‘break’ out of the loop. Otherwise, I use ‘parseInt’ to then take the numerical values the user enters (which they can enter on one line, with a space in between) and assign them to ‘row’ and ‘column’ respectively. Consequently, I pass these variables as parameters to ‘hit’ through ‘board’ to test if the result would be true. If it is, then the hit was on target. Otherwise, it was not. This loop continues until the user enters “exit”.

In ‘Board’, ‘hit’ contains a for loop and takes a temporary variable, using the parameters previously passed to it, to make a temporary ‘Part’ object called ‘partSupplied’. I also create a boolean variable ‘partMatches’, initialised as false. I go on to use a for-each loop for each of the Ship objects in my ‘ships’ ArrayList (which is a private ArrayList in my ‘Board’ class) testing whether the result of ‘hit’ from my ‘Battleship’ class, for each one, does not return the integer value 3. Although strange at first, this is actually because in my ‘Battleship’ class, I decided that I wanted to distinguish between three results (either part of the ship has been successfully destroyed, the part of the ship targeted has already been destroyed, or the hit has missed). For this reason, in ‘Battleship’, my ‘hit’ method is of type integer and returns either 1, 2 or 3 – again, purely to increase functionality if, for example, I wanted a separate display message for each of these scenarios. Back in my other ‘hit’ method, in ‘Board’ as to carry on from earlier, if the result of ‘hit’ in ‘Battleship’ is not 3, then this means that the hit was on target and, unless in the case of minesweeper (which has its own additional condition causing a 50% chance of withstanding any damage), the hit either destroyed a piece, or the piece hit is already destroyed. At this point in my code I set ‘partMatches’ to true and simply pass ‘row’ and ‘column’ again through to ‘hit’ in ‘Battleship’, which actually sets the value of ‘isDestroyed’ for each of the parts in a ship – that is, it manages the destruction officially of parts being targeted. With this done, within my ‘hit’ method in ‘Board’, I test whether ‘partMatches’ is now true since if it is, then the row and column passed to the method were correctly on target. If true, I print “Hit!” and return true. If partMatches is not true, the attempted hit was not on target – I print “Miss!” and return false.

In terms of my actual ships and their type, I use my class ‘Battleship’ as a superclass for my other three ship types, the subclasses ‘Cruiser’, ‘Frigate’ and ‘Minesweeper’. The constructor in each of these classes takes advantages of the first constructor in ‘Battleship’ – however each takes only one

parameter, for ‘row.’ This is because the number of columns taken up by each of these ships depends solely on which type of ship it is, and this will not change – for example, a minesweeper always takes up 2 columns. Ships are constructed within these constructors using the keyword ‘super’, passing the row value to the constructor as well as a constant integer column value as previously discussed (i.e. ‘Cruiser’ always taking up 4 columns).

‘Minesweeper’ is the subclass of ‘Battleship’ in which I actually override the ‘hit’ method. Having already imported ‘Random’ from the Java class library in this class, I use an object I make of ‘Random’ called ‘rnd’ and determine a 50% chance using rnd.nextInt(2)., a line of code which returns pseudorandom out of only either 0 or 1. If 1, then I proceed to call the original ‘hit’ method (which deals with fields such as ‘isDestroyed’) which is in ‘Battleship’, passing the same row and column already passed to the overridden ‘hit’ method in ‘Minesweeper’. Otherwise, if 0, I use if statements to still determine whether, had it been a 1 instead, the result of ‘hit’ in ‘Battleship’ would return 1, 2 or 3. This, as stated earlier, simply helps me to be more exact in my print statements – i.e. “The shot was on target - and would have hit for damage - but the resilient minesweeper took no damage this turn” – or, similarly: “the shot was on target, but that part of the ship has already been destroyed and the minesweeper takes no damage this turn”.This helped me particularly in developing and debugging my program, and I am particularly proud of, and impressed with my decision to make ‘hit’ an integer type allowing me to create a three-way rather than simply using a Boolean one. It is not something I had ever previously thought of before.

Something I struggled to implement in my solution, and which I first implemented and later re-implemented in a much nicer way, was my ‘toString’ methods seen in my ‘Battleship’ class and ‘Board’ class. Originally, in printing my board to the screen my ‘toString’ method in ‘Board’ would initially return ships.toString() – printing the String representation of each ship in my ‘ships’ ArrayList. The problem here was printing it as an ArrayList, causing extra square brackets to be put on the very outside of the contents. Similarly, in ‘Battleship’ my ‘toString’ method would initially return ‘parts.toString’; the String representation of each part in my ‘parts’ ArrayList – again as a whole ArrayList, causing extra square brackets through my grid display and, consequently, completely altering the display of my board, making it appear non-uniform. This was solved by instead using a for loop in each of the two separate ‘toString’ methods, appending each new item in the ArrayList onto a String variable and then returning it, thereby escaping returning the contents as a whole ArrayList and thus evading unnecessary spare square brackets.

In terms of implementing a part of my code again to make it more efficient – this was done after first realising I could extend ‘Battleship’ in creating my ‘Minesweeper’ class – in my initial implementation of the program, I copied and pasted all of the code for the ‘hit’ method from ‘Battleship’, into ‘Minesweeper’ and added the code for the 50% chance to override it. However, after becoming more used to subclass/superclass concepts, I made this more efficient by overriding ‘hit’ with only the 50% chance condition, and then simply referring back to the ‘hit’ method in ‘Battleship’ in the case of my 50% chance outcome being a 1 rather than 0.

Something which I removed in my final implementation of my program was two ‘else if’ statements providing extra user friendliness in the form of extra information (as print statements). This appeared near the end of my overridden ‘hit’ method in ‘Minesweeper’. This can be seen still in my pseudocode, from the planning stage of my solution which still shows how this would have been previously implemented had it not caused any issues. The reason these were removed entirely was because within each of the ‘else if’ statements was a condition involving ‘super.hit(row, column)’ returning a value equal to an integer of either 1 or 2. The intention of this was to be able to compare against these values to ultimately display a suitable message for the scenario, in higher detail. The

calling of this method, however, perhaps somewhat foolishly guaranteed that hit was always, no matter what called through the ‘Minesweeper’ class, when it is only supposed to happen under a 50% chance. Consequently, this was removed.