Object Oriented Program Design 110

Assignment Specification

Final Assignment Submission: 12.00 midday, June 2nd 2014

This assignment contains three sections:

1. The assignment specification.
2. Appendix A: The assignment cover page template. Assignments submitted without the completed cover sheet will not be marked.
3. Appendix B: A proposed time schedule for successful assignment completion.

Logistics

This is not an assignment which can be put aside until a few days before it is due. You are required to perform assignment based tasks on a weekly basis. Your final mark will be directly coupled to:

- Your ability to ensure that progress on the assignment is made every week and that the weekly reading tasks are successfully completed.
- Your ability to communicate and to read and comprehend text based material.
- Your ability to demonstrate an understanding of how to design, implement and test an object oriented software application.
- Your ability to pro-actively seek assistance when required.

Marking Strategy.

The Pseudo code algorithm and Java implementation will be marked. The marks for your reading interviews will then be converted to a weighting in the range 0.0 to 1.0. This weighting will then be applied to the mark calculated from marking your pseudo code and Java code. The result is your assignment mark. You will be interviewed on the first eight chapters of the pdf file text book in the practicals occurring in the weeks indicated in the schedule contained in the unit outline. Please keep checking this timetable so that you are aware of when the reading interviews will occur. If you miss your practical in a week which has reading interviews then contact me immediately for an alternative. Note immediately means the second you realise you have missed your interview not three weeks later.

Submission Requirements

The assignment must be handed in printed (not handwritten) form. Electronic copies of all assignment documents must be kept in your Curtin user account. All assignment data should be kept in a directory called OOPDAssignment. This directory should be located in your home directory. The assignment cover sheet should follow the template provided in appendix A and all the information specified must be supplied and be correct. You must also sign the declaration on the cover sheet. Failure to conform to these requirements will result in your assignment not be accepted and marked.

Each assignment submission should be made in hard copy form and should contain:

- A completed cover page (as specified in appendix A).
- A complete set of pseudo code and Java for the algorithms and code implementation for the software. Note the Java code must be compliant with the Dept of Computing Java coding standard (available from Blackboard) and must be fully documented. The Java code must be a valid implementation of your supplied pseudo code.

Java Application Programmer Interface (API)
Java comes with a fairly impressive library of classes which, for a developer, is incredibly useful. Unfortunately this makes an assignment in a unit such as OOPD110 very difficult because most tasks can be done via this library. For this reason you may **not** use any of the Java API classes other than:

- Those already used in the worksheets (e.g. System.out)
- Those mentioned in Appendix B.

You may also use the ConsoleInput class as you have been in your practicals. For the purposes of this assignment you may assume that when inputting a filename that the name consists of a collection of alphanumeric characters, followed by a period, followed by 1 or more alphanumeric characters. This means you can use the ConsoleInput.readWord method for this task. **Note that ConsoleInput.readLine() does not work correctly under windows so do not use it.**

The Game
You must design in pseudo code and implement in Java a game for playing Battleships. Traditionally this game is played by two players. Each player has a map of the battle area divided into a grid. Each player places their collection of ships on the grid, keeping their map hidden from the other player. Ships cover more that one square of the map grid. The two players then take turns at trying to hit the other player's ships. When taking a turn the player calls out a grid location. The other player then checks their map and if the grid location contains part of one of their ships then they call out hit, otherwise they call out miss. When all of the grid locations covered by a ship have been hit then that ship is considered to be sunk. The first player to have all their ships sunk is the loser of the game.

Your assignment is to write a Battleships game in Java. However to reduce the complexity to an appropriate level there are constraints. The software will act as defender and the player is trying to sink all of the ships. The map containing the grid with ships placed on it is a text file which is created by hand before the application commences.

The user is asked for the name of the data file which contains the map. The map is read from the file into memory. The game is then played by the user continually inputting a grid location and the software determining if a ship has been hit or not. The result is output to the user. This process continues until all the ships have been sunk then the application outputs the number of hits and misses scored by the user. In a normal game of Batteships, at each turn, the user would only be told whether or not they hit a ship or missed a ship. In this version the user can either see a map which just shows the hits (but not the ships) or one showing the ships as well. You will need to see the ships in order to determine if your application is working correctly.

The Input File
The software applications reads the map data from an input file. All maps must conform to the following specifications:

- The first line of the input file contains an integer representing the size of the map.
- The remaining lines contain each row of the maze as characters.
- A map is represented by a grid of characters where a hash symbol (i.e. '#' represents part of a ship, a '-' character represents empty ocean and a ship hit is indicated by a capital X character (i.e. 'X').
- All mazes are square and have a minimum size of 10 characters and a maximum size of 20 characters.
- All ships are exactly 4 characters in length and 1 character in width.
- Two ships cannot occupy the same grid.

- Ships are always placed vertically or horizontally.
- Grid positions are referenced by specifying a row, column position. Rows and columns are numbered from zero. e.g. if the map size was 12 the rows would be numbered from 0 to 11 and so would the columns – the top left hand corner would be position 0,0 and the bottom right hand corner would be position 11,11.

An example map is show below:

```
20
--------------------
--------------------
--#-----------------
--#-----------------
--#-----###--------
--#-----------------
---------------####
----------#--------
----------#--------
-------#---#--------
-------#---#--------
-------#------------
-------#------------
--------------------
---###-------------
--------------------
#-------###--------
#-------------------
#-------------------
#-------------------
```

Your algorithm must check that a data file can be opened successfully but once opened, you may assume that the map data within it is valid. Note you must make your own map data files for testing your algorithm. It is recommended that you devise several maze sizes ranging from 10 to 20 and start with the above example (you can download it from blackboard).

OOPD-110 does not teach file access in Java so a small Java library has been provided for your use (TextFile.java which can be downloaded from Blackboard). All file access MUST be done via the Java code in this file. **You are not permitted to make use of any of the standard Java API for file access. You are also not permitted to make any changes to the code in this file.**

The TextFile.java file has been documented and all students should print out the file and go through and understand the function of each method in the file BEFORE they attempt to make use of it in their assignment. To test your knowledge you should develop some test programs which do things like copying the file out to the screen.

A two dimensional array is also required and this is also not covered in OOPD 110 (if you do not know what an array is don't worry. Think of a 2 dimensional grid arranged in rows and columns). Therefore a super class called Map.java has been provided to encapsulate the required array functionality. As with TextFile.java you are required to use this file but are not permitted to make any changes to it. A position in the maze is specified by stating which row and column of the maze you are referring to. As previously stated, rows and columns are numbered from zero to the size of the map minus one. For example if your map was 12 in size then it would have 12 rows and 12 columns. The rows and columns would be numbered from 0 to 11. The position in the top, left corner of the maze is row

zero, column zero. The bottom left hand corner position would be row 11, column 0, the top right hand corner position would be row 0, column 11 and the bottom right hand corner would be row 11, column 11. When specifying a position it is always the row first followed by the column (e.g. row, column).

You must have the following classes in your design. You should have other classes as well but these ones must be present and must fulfil the specifications below:

Class TextFile: used for file access and provided for you to use. Take a copy of this code from blackboard. There is no need to provide pseudo code for this class. **You must not modify this code in any way.**

Class Map: used for administering the map. The constructors are responsible for setting the size of the map but not filling it with characters. See above description of how a map is represented and also check the constructors, accessors and mutators so you know how to deal with this class. Take a copy of this code from blackboard. There is no need to provide pseudo code for this class. **You must not modify this code in any way.**

Class BattleShips: This class must inherit from the class Map. Its sub class functionality must provide a means of:

- Reading the map from an input file.
- Playing the battleships game as previously described.
- Displaying the maze after each move. While not part of the game this functionality will be needed in order for you to establish your application is working correctly. After each turn the user should be asked if they wish to see the map. If they indicate they wish to do so then the map should be displayed.

You must design this class in pseudo code and implement it in Java.

Class Position: used store a map grid position. You must design this class in pseudo code and implement it in Java.

Object Orientation Issues

In addition to the previously mentioned class, others may be required. Students should give careful thought to the best arrangement of classes for the application. Marks will be allocated for your choice of classes as well as for the overall functionality of your algorithm and Java implementation.

Important Marking Information

Please note that your algorithm will be assessed by examining the pseudocode, not the Java. Your Java code will be assessed by its suitability as a valid implementation of your pseudocode.

Appendix A

Object Orientated Program Design 110
Assignment Cover Sheet

Semester One
2014

Student Number: _____

Family Name: _____

Other Names: _____

I declare that:

1. I am aware that use of work done by others without declaring where the work came from IS plagiarism.
2. This assignment is all my own work (other than the code supplied by the unit controller).
3. Curtin University has a Student Misconduct Policy and, should I use the work of another in my assignment then I will be in breach of this policy.

Student Name (Print):_____

Student Signature:_____

Date:_____

Appendix B

Development Schedule

Often students make the mistake of waiting until everything required for an assignment has been covered before they commence work on their assignment. This is not possible in the case of this assignment. Firstly the reading part of the assignment occurs throughout the semester.

In terms of the programming task I suggest the following schedule:

- By week 3 have an understanding of what the assignment is asking you to do. You will not know how to do it but be very familiar with this document. use a highlight pen to emphasis bits that you think you will need to refer back to in the future.
- By week four, you should be able to simulate the program by hand and therefore identify the main steps involved. Hint: Use a piece of graph paper as your map and get used to refering to positions in the map grid by specifying rows and columns.
- By week 6 design in pseudocode and implement in Java, a program which will input a file name, open the file and output its contents to the screen.
- By week 7. Modify the above program so that it can read a map data file and output its contents to the screen.
- By week 8, design an algorithm which will open a file, construct a map and input the map data into the map object.
- During weeks 9-12 you have most of the bits and pieces done to complete the assignment so now incorporate the inheritance, finish it and test it.