CHC6186 Advanced Object-Oriented Programming

Coursework

For this coursework, you will produce in Java two versions of the game Numberle. One version will have a Graphical User Interface (GUI) and the other version will have a command-line interface (CLI). The GUI version will be constructed according to the principles of Model View Controller, and the CLI version will use the same model. The two versions will from now on be called the GUI version and the CLI version.

Learning Outcomes

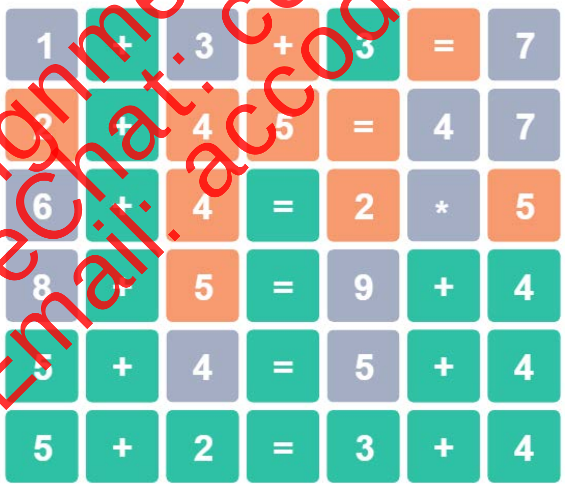
This coursework will assess the following learning outcomes.

- Create a software artefact by applying the methodologies of advanced object-oriented programming to a requirements specification
- Consult on-line code libraries to find the classes and methods most appropriate for solving a problem
- Create appropriate documentation that clearly communicates the intended behaviour of a program

This coursework is worth 50% of your module mark; the remaining 50% comes from your exam.

How to Play Numberle

Numberle is a mathematical equation guessing game where players must accurately guess a randomly generated equation within six tries.[1] Players enter their own equation, aiming to match the target equation. In total, players have 6 attempts to guess the target equation. When calculating, players can use numbers (0-9) and arithmetic signs (+ - * / =).
For this coursework, the length of the mathematical equation is fixed at **7 characters**. (However, the character number of the link *numberle.org* is originally 8, but you can change it to 7 characters by clicking the top left setting button) In each attempt, the player enters their own correct equation to find out what numbers and arithmetic signs are in the equation. If the number or sign is in the equation, but in the wrong place, it will be highlighted in orange. If it is in the exact place, then it will be highlighted in green. If there is no number or sign in the equation at all, the color will be gray. In this coursework, arithmetic expressions are evaluated using BODMAS. BODMAS stands for "Brackets, Orders (exponents), Division and Multiplication, Addition and Subtraction." This means that operations within brackets are performed first, followed by any exponents, then division and multiplication (from left to right), and finally addition and subtraction (from left to right).



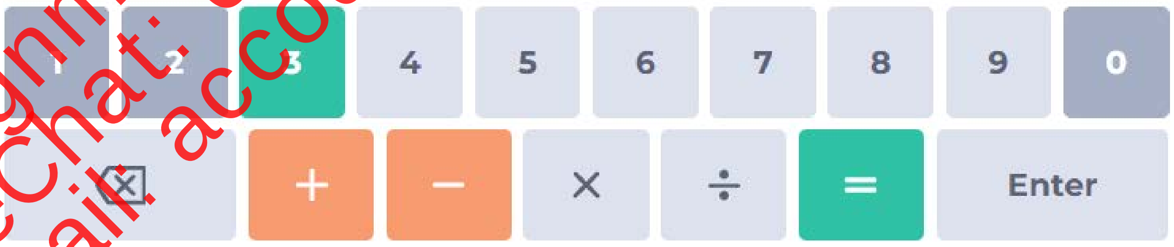
The website is implemented in Javascript. Any attempt to submit Javascript will receive a **mark of zero** and any Java based on the website's Javascript will be treated as **plagiarism** in the normal way. The website colours may be used.

[1] https://numberle.org

Functional Requirements

For greater clarity, the description of the GUI and the CLI versions of the game can be summarised in the following list of functional requirements.

FR1	For the GUI version, a confirmatory message or a message box should be displayed to indicate whether the player has won (guessed the mathematical equation) or lost (run out of guesses), even though the game status is clear from the tile coloring on the last filled row.
FR2	For the CLI version, a confirmatory message indicating the player has won or lost is required.
FR3	The behaviour of the program shall be controlled by three flags: • One flag should, if set, cause an error message to be displayed if the equation is not valid; this will not then count as one of the tries. • Another flag should, if set, display the target equation for testing purposes. • A third flag should, if set, cause the equation to be randomly selected. If unset, the equation will be fixed.
FR4	Both GUI and CLI versions of the program should allow players to input their guesses for the mathematical equation, consisting of numbers and arithmetic signs.
FR5	The Model should load a list of valid equations from a fixed location (from one provided file equations.txt). This list will serve as potential guesses for the player.
FR6	The GUI should display a keyboard in which digits or signs are displayed in dark grey if it has been revealed that they do not occur in the mathematical equation, green if a correct location of a digit or a sign has been found, and orange if the digit or sign has been guessed but never at the correct location. See below for an example; this functionality is like the GUI shown on the website. The CLI should indicate available digits or signs by listing them in four separate categories in a certain order.
FR7	The GUI version should have a button to ask for a new game which will be enabled only after the first valid guess has been made. This is not required for the CLI version.



Non-functional Requirements

The following non-functional requirements also apply

NFR1	The GUI version and CLI version should be two separate programs ie there should be two files each with a main method in them and which file is run determines which version activated.
NFR2	The GUI version must be constructed according to the principles of MVC, as restated below. Because of this requirement, code that belongs in the View but is placed in the Model will usually not be counted towards the marks for the View. Similar rules will apply for other misplaced code.
NFR3	The CLI version will use the Model part of the GUI version directly without using the View or Controller; nor should it define a new view or controller.
NFR4	The code must be documented with asserts, unit testing, class diagram, comments as described below.
NFR5	The code must be of good quality as described in the marking scheme below.
NFR6	The flags mentioned in FR3 should be in the Model. It is not necessary for them to be changeable at run time.
NFR7	The model should also have a constant indicating the number of allowable guesses.

Marking Scheme (See rubric as well).

Model. This should have an interface designed to be convenient for the Controller, View and JUnit class to use with no superfluous public methods, no references to two classes and contain no GUI code. It may consist of several classes but there must be a class called Model or similar that provides the interface and this class should extend Observable. File reading should also be done in the Model. A high mark will be earned for a Model that implements all the required functionality and respects all these constraints. A pass mark will be earned for a Model that implements only some of the required functionality or fails to respect these constraints.	20%
Controller. This should forward only valid requests to the Model, querying the Model if necessary to find out if the request is valid, and must also enable / disable buttons as described above in the functional requirements. It must have no GUI code, though it may send messages to the View. A high mark will be given to a controller that respects all these constraints and a pass mark will be given to a controller that respects only some of them	10%
View of GUI version using the Swing framework. It should implement Observer and therefore have an update method that is called when the Model changes. This will be marked according to how many of the functional requirements have been met. A high mark will be given to a view that implements all the requirements and a pass mark will be given to a view that implements only some of them.	10%
CLI version of the program, using the Model.	10%
Specification of Model with asserts. This should include invariants for the class as well as pre and post conditions for each public method in the model. This will be marked according to how many of the relevant conditions are included and whether the ones that are included are correct. Partial credit will be available for describing them in English. A high mark will be given to a specification that includes all the relevant constraints. A pass mark will be given to a specification that includes only a few of them.	10%
Unit testing of the Model in JUnit. There should be three tests, significantly different from each other. You should explain in comments the scenario ie the situation you are testing for. You should use write (and then call) methods for the Model that set it into the state desired for the test. It should be easy to see what state the Model is being set to by reading the code for the unit tests. A high mark will be given to significantly different tests	10%

that are easy for the marker to interpret. A pass mark will be given to unoriginal second or third tests or to three tests that are difficult to understand. Your Model may use a separate Board class but the testing should be of the Model class and the specification should be applied to that class also.	
Use of the code quality practices described in Lecture 1, plus the additional practices of light relevant commenting and correct formatting. Short elegant programs are preferred, and code smells are to be avoided. Note that high marks for this category will only be possible if the GUI fulfils most of the requirements. A high mark will be awarded if all the practices are observed and a pass mark will be awarded if only some of them are.	10%
Class diagram. This should show how the Model, View and Controller are related to each other, as well as how they interact with library classes such as Observable. Simplicity and clarity will be reward. It will be marked according to its accuracy as a representation of the program. A high mark will be awarded for an accurate diagram and a pass mark will be awarded for a less accurate diagram.	10%
Video presentation that shows you displaying the code and using the program. It will be marked according to timing, presentation and how well you show that you have met the FRs and NFRs in both versions.	10%

Submission

Requirements

1. Your submission should contain **three files** (.pdf, .zip, and .mp4).
2. The first file is a .pdf document with screenshots of the implementation (Java code), testing, and design with a class diagram.
3. The second file is a .zip file with the Java project.
4. The third file is a .mp4 video that is less than 1 GB. If the video is not viewable, it will not receive marks. The video must be a maximum of **five minutes** long during which you must display most of the relevant functionality and refer to your code. Any recording software can be used so long as it captures your screen and your voice.
5. Additionally, you are required to regularly upload your code to **GitHub** as per the university counterpart's requirement. Provide the GitHub repository link in the PDF document.
6. The PDF document is the version that will be marked, but the .zip and .mp4 are requested so that we may run the code.

File Naming Convention

You must save the files with the following names:

- {YourStudentNumber}-coursework.pdf
- {YourStudentNumber}-coursework.zip
- {YourStudentNumber}-coursework.mp4

For example:

- 202007081314-coursework.pdf
- 202007081314-coursework.zip
- 202107081314-coursework.mp4

Submission Deadline:

You must upload from the student website (student.zy.cdut.edu.cn) before **17:00**, **May 6th** (Monday).

Some students will be selected to give a Zoom presentation, after the exam period. If you are asked to give a Zoom presentation then you must do so.

Formative Feedback

We are giving you the opportunity to receive feedback on the design of your program. To receive this feedback, you need to upload a detailed UML class diagram of your code to student website before **17:00** on **Friday March 25th**. As this is a formative feedback deadline, it will not be possible for you to seek deadline extensions. You will be given a short amount of written feedback on your design within a week. The Week 5 teaching session will go through a worked example in order to help you produce the class diagram.

The class diagram should have all methods and attributes showing. In addition, you should indicate which methods call which other methods. A class diagram with insufficient detail or syntactically nonsensical or not realisable as an actual Java program will make it more difficult for us to give you feedback and will receive a low mark if submitted with the final report.

Academic Conduct

This is an individual piece of work and you will have to work on your own and submit your own original attempt at the assignment. Any code that has been copied from any source (e.g. Stack Overflow, online tutorial, textbooks, other students etc.) must be properly referenced to avoid any suspicion of plagiarism. Refer to the Module Handbook for further information on this. If you need help you can always ask for advice and guidance from the module leader by email; online sessions can be arranged for further clarification.

Rubric

The work shall be marked according to the following rubric.

	D	C	B	A
Model	only basic functionality implemented or slightly more than basic but references to View or Controller or superfluous methods	no superfluous methods and no references to View or Controller but only the basics of functionality implemented	no superfluous methods and no references to View or Controller but only the basics of functionality implemented	convenient to use with no superfluous methods, all required functionality and no references to View or Controller, extends Observable, calls setChanged and notifyObservers
Controller	zero of the requirements: only valid requests, querying Model first, enables/disables buttons without GUI code	one out of only valid requests, querying Model first, enables/disables buttons without GUI code	two out of only valid requests, querying Model first, enables/disables buttons without GUI code	only valid requests, has references to both Model and View, converting UI interactions into methods to change the Model, querying Model first, enables/disables buttons without GUI code
GUI View	no view update method or update method implementing very few of the FRs	update method in view implementing some of the FRs	update method in view implementing most of the FRs	update method in view implementing all the FRs, uses Swing, has Model and Controller as attributes, displays board and allows Controller to change the view e.g. enable/disable options, implements Observer and calls addObserver
CLI class	CLI version implementing very few of the FRs	CLI version implementing some of the FRs	CLI version implementing most of the FRs	CLI version implementing all the FRs, using same Model as the GUI version, but no Controller and is demonstrated on the video
Specification of Model with asserts	a few pre/postconditions described in English	suitable pre/post conditions for most public methods but in English	suitable pre/post conditions for most public methods expressed in some logic	suitable pre/post conditions for all public methods and class invariants all expressed as statements of formal logic
Unit testing of Model with JUnit	one test with the scenario poorly described or not at all	tests all essentially similar or only one or two or scenario being tested poorly described	third test not significantly different or scenario being tested not described with sufficient care	three significantly different tests of the model with all scenarios exactly described and with all inputs satisfying the preconditions
Code quality practices	most code quality practices not observed	some code quality practices observed but many not	most code quality practices observed but some clearly not	all code quality practices observed including light correct commenting, suitable identifier names (constants, methods, classes etc) in appropriate cases, indentation, lack of code smells (long methods, repeated code, lack of modularity)
Class diagram	Inadequate class diagram with serious mistakes in attributes and relationships between classes	Adequate class diagram with mistakes in both attributes and relationships between classes	Good class diagram with only a few mistakes in attributes, visibility or relationships between classes	Excellent class diagram with all attributes indicated with correct visibilities and correct relationships between classes all shown
Video Presentation	Very poor presentation with insufficient coverage of FRs and NFRs, poorly presented and overly long	Passable presentation covering FRs or NFRs or well-presented or at least appropriate length	Quite good presentation but missing some details of FRs and NFRs or poorly presented or overly long	Excellent presentation with full explanation of most FRs and NFRs, referencing the code, well presented and within time limit