CFS2160 Software Design & Development -
Assessment 2 – Writing a Full Program

Written By : Dr Gary Allen
Date : Last Update September 2017
Handed to Students : Week 3 Term 2
Hand In Date : 12.00 Noon on Thursday 22nd March 2018 (week 10 term 2), with demos taking place throughout weeks 23 and 24
Returned to Students : End of Year

Aims
To develop skills in modelling using UML
To develop skills in defining classes
To develop skills in writing programs
To develop skills in testing programs
To demonstrate the need to document software

Outcomes to be Assessed
This assignment addresses all learning outcomes for the module.

Problem Statement
You are required to design, implement, test and document a Java program (set of classes) for ***one*** of the following problem specifications.

Problem 1 - Who wants to be a Zillionaire?
The problem is to define classes suitable for playing a game of 'Who Wants to be a Zillionaire'. This game is a cross between the successful TV quiz shows 'Who Wants to be a Millionaire' and 'Mastermind'. Players choose a category on which to answer questions. Questions are of a multiple-choice nature with the answer being one of four possibilities. If the player answers the question correctly an amount of 'money' is credited to that player. Questions have a difficulty level according to the amount of 'money' players are awarded for answering that question correctly. The questions are asked in ascending order of difficulty and, therefore, value. Players take turns to answer questions. If a player cannot answer a question, help is available as follows:

- Ask the Public. This involves a simulation of each possible answer being given a rating according to the likelihood that it is the correct answer. Ratings for all four possible answers must add up to 100% in total.
- Half-and-half. Two of the possible answers are removed leaving only the right answer and one wrong answer.

Players are only allowed to use each help facility once per game. If a wrong answer is given that player is out of the game. The winner is the player with the most money credited at the end of the game.

To obtain a bare pass your game should have most of the following functionality:

- Allow two players to play the game
- Include a selection of questions that players could be asked
- Allow a player to choose the category of each question (for simplicity have three categories, one of which should be general knowledge)
- Display a question at random from the available questions
- Provide a facility for the player to enter the answer to each question
- Inform the player whether the answer is correct, or not

- Keep track of the amount of ‘money’ credited to each player
- Implement the help facilities
- Allow a player to win

To gain a higher mark you will need to implement some extensions to your game. Some suggestions might be:

- Allow more than two players to play the game.
- Facility for easily adding extra questions.
- Add another help facility, e.g. ‘Give us a Clue’.
- Ensure that the same question is not asked twice in any one game.
- Include the use of sound or animation to make the game more interesting.
- Display the ratings for ‘Ask the Public’ in graphical format e.g. a bar chart.

Problem 2 - Ten-pin Bowling Alley Scoring System Simulation

A local ten-pin bowling alley would like a simulation of the scoring system for training and development purposes, and also to show to customers who are new to the game. Ten-pin bowling is a game played by two or more players and consists of a ‘lane’ with ten ‘pins’ at the end (a ‘pin’ is like a skittle). The object of the game is to bowl a ball towards the ‘pins’ in order to knock down as many ‘pins’ as possible. The bowler is allowed ten frames in which to knock down ‘pins’, with each frame being composed of up to two rolls. Players score a point for each ‘pin’ they knock down in any one frame. If the bowler scores a ‘strike’ (all 10 ‘pins’ are knocked down at the first attempt) or a ‘spare’ (all 10 ‘pins’ are knocked down in two attempts) bonus points are awarded in addition to the 10 scored. The number of ‘pins’ the bowler knocks down in the following frame is used to calculate the player’s bonus. A more detailed explanation of the game and its scoring system, including how to calculate the bonus points, can be found at: http://en.wikipedia.org/wiki/Ten-pin_bowling.

To obtain a bare pass your ten-pin bowling scoring simulation should:

- Allow two players to play
- Provide a method of signalling that a ball is to be bowled (a button will do)
- Enable a player’s score to be randomly generated for each ball within a frame
- Provide a display of the number of ‘pins’ knocked down by each ball
- Keep track of the player’s score and display it on the screen
- Handle spares and strikes correctly (including in the 10th frame)
- Inform the players of the winner of the game

To gain a higher mark you will need to implement some extensions to your system. Some suggestions might be:

- The facility to have more than two players, but a maximum of eight
- The facility to have more than one lane, each with two to eight players
- Provide a more sophisticated display of the number of ‘pins’ knocked down by each ball
- Allow players to customize the view of the score sheet, such as being able to input a name for each player
- Improve the method of generating a player’s score
- Allow players to place an order for drinks and snacks from the café

Getting started
You will need to identify the use cases and define classes to represent the problem you have chosen to implement. In addition, your system must include a graphical user interface showing the state of the system at all times. You should try to design your user interface and the classes that you will need before starting coding. When you are ready to begin coding start with a basic program that has a very simple user interface and some of the simpler functionality to obtain a pass. When this is working properly improve the user interface and try implementing some of the more complex functionality. Finally, add one or more extensions to gain a better mark.

Remember that the extensions given are just suggestions and you do not need to implement them all (or indeed any of them, if you can think of some of your own) to gain a high mark. However, there is no limit to the extensions you can implement; the only limit is your imagination (and programming skills!).

Hand In
Formative Feedback
You are invited to submit for formative feedback a one-page use-case diagram and a one-page class diagram of the design of your software. These should be as complete as possible and so the use-case diagram should show the actors and associated use cases and the class diagram should show fields, methods, visibility etc., as well as relationships between the classes. Please bring your diagrams to any practical class before the end of week 17 (i.e. before Friday 16th February).

Week 22: Thursday 22nd March, 2018
You must submit:
1. An electronic copy of your classes (including test classes). Electronic copies should be within an IntelliJ or Eclipse project. All code should be professionally written (including comments and indentation, use of sensible variable names, etc.), and will be assessed for:

- correctness
- robustness
- appropriate use of language constructs
- style (commenting, indentation, etc.)
- design (consideration given to cohesion, coupling, maintainability)
- extensions (extra marks for difficult/original extensions)
- testing in the form of test classes

The code counts for 50% of the marks.

2. An electronic report that includes:

- The final design documentation for your system, to include *as a minimum* a one-page use-case diagram showing the actors and associated use cases and a one-page class diagram of the design of your software. Note that this should be an updated version of the diagrams you presented for formative feedback, improved to take into account any feedback received or any changes made as the software has been implemented. This design documentation should be produced using the Visual Paradigm UML modelling software. You should also include brief notes (maximum of 500 words) to explain your design. This component will count for 30% of the marks.
- A test plan showing the application tests that you have used. This should be supported by some screen shots that show evidence of the testing. This should typically take 2 – 3 pages. This is worth 20% of the marks.

How and where to hand in
Please create a folder labelled with your student ID and name (e.g.: u1234567_FirstName_LastName). Place your IntelliJ or Eclipse project in this folder with your report. Compress the folder using standard compression software and submit the compressed file to UniLearn. There will be a link for this under the Assignments tab.

Week 23 & 24: Demonstration
All students must demonstrate their software to their tutor in either week 23 or 24. Failure to demonstrate your software will result in a mark of zero being awarded for this work. A schedule will be available via UniLearn for the demonstrations. You should expect to show your tutor how your software works and answer any questions that are asked.

Assessment Criteria
This assignment comprises 50% of the total marks for the module.

The modelling will be judged and assessed on:

- The quality of the design of the software;
- The accuracy of the use of UML notation; and
- The completeness of the diagrams.

The software will be judged and assessed on the quality, and adherence to industry standards of:

- The computer program;
- The testing of the program; and
- The documentation of the code.

Group Component
There is no group component, this is an individual piece of work. All work must be your own. Collusion is cheating and will be dealt with severely. If I have any doubts as to the originality of any work submitted I will report this to the Academic Conduct Officer who will hold a full investigation.

Resources Required
Access to a computer with IntelliJ, Eclipse, and Visual Paradigm available.