

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**SCHOOL OF ENGINEERING AND TECHNOLOGY**

**FINAL ASSESSMENT FOR THE BSC (HONS) INFORMATION TECHNOLOGY; BSC (HONS) COMPUTER SCIENCE; BACHELOR of SOFTWARE ENGINEERING (HONS)YEAR 2**

**ACADEMIC SESSION 2024; SEMESTER 3**

**PRG2104: OBJECT ORIENTED PROGRAMMING**

**Project DEADLINE: Week 14**

**INSTRUCTIONS TO CANDIDATES**

# 

# This assignment will contribute 50% to your final grade.

* This is an individual assignment.

**IMPORTANT**

# The University requires students to adhere to submission deadlines for any form of assessment. Penalties are applied in relation to unauthorized late submission of work.

# Coursework submitted after the deadline will be awarded 0 marks

**Lecturer’s Remark** (Use additional sheet if required)

I.....Wong Yee Ling...... (Name) ...21090857....std. ID received the assignment and read the comments...........ling............ (Signature/date)

**Academic Honesty Acknowledgement**

“I .....Wong Yee Ling.....(student name). verify that this paper contains entirely my own work. I have not consulted with any outside person or materials other than what was specified (an interviewee, for example) in the assignment or the syllabus requirements. Further, I have not copied or inadvertently copied ideas, sentences, or paragraphs from another student. I realize the penalties *(refer to page 16, 5.5, Appendix 2, page 44 of the student handbook diploma and undergraduate programme)* for any kind of copying or collaboration on any assignment.”

…......ling............ (Student’s signature / Date)

**GitHub Account:** <https://github.com/bull4shit>

**Final Project Link:** <https://github.com/bull4shit/Hangman>

**UML Diagram: Class and Object Relationships**

**A diagram of a computer

Description automatically generated**

The Hangman program can be described using various UML diagram to show its structure and behavior. The primary class contains numerous attributes such as ‘words’, which is a list of possible words in the game, ‘word’, which stores the current word to guess, ‘guessedWord’, an array to show the player’s progress, ‘attemptsLeft’, to keep track of the remaining attempts left, ‘guessedLetters’, a set of characters representing the letters already guessed. Other than that, it has various UI components like “wordLabel”, “messageLabel”, ‘hangmanImageView’ and ‘keyboardButtons’, which are instances of classes like ‘Label’, ‘ImageView’ and ‘Button’.

The ‘Label’ class represents text elements in the GUI, such as ‘wordLabel’ and ‘messageLabel’, it displays the current guessed word and prompt message to the player. The ‘imageView’ class is used for ‘hangmanImageView” which displays the hangman images that change based on the player’s incorrect guesses. The Button class is used to create interactive buttons, including the on-screen keyboard represented by keyboardButtons, where each button corresponds to a letter of the alphabet. The buttons in the game are responsible for initiating operations, particularly the handleGuess method in HangmanGame, which handles the player's guesses.

The Object Diagrams capture the state of objects during runtime and show individual instances of these classes. For example, it might show the current state of the hangman picture or wordLabel could display a succession of asterisks signifying the hidden word. This graphic shows the object states as they are at a specific point in time during the game.

The Sequence Diagram will show the dynamic interaction between these objects during the processes of the game, such as when a player guess a letter. When the player presses a button corresponding to a letter, this triggers the handleGuess method in HangmanGame. The game then checks whether the letter has already been guessed, updates the guessedWord if the guess is correct, or decreases attemptsLeft and updates the hangman image if the guess is incorrect. The game then updates the UI components, such as wordLabel, to reflect the new state. If the player successfully guesses the word or runs out of attempts, the game will display an appropriate alert using the Alert class, either congratulating the player or informing them that the game is over.

As the game progresses, such as when a player guesses a letter, the Sequence Diagram will display the dynamic interaction between these elements. Pressing the button corresponding to a letter in HangmanGame will activate the ‘handleGuess’ technique. After that, it determines if the letter has been guessted before, adds the correct guess to the guessedWord, or reduces the number of attempts. If the estimate is wrong, it updates the hangman image. Moreover, the game modifies the user interface elements, like wordLabel, to match the changed state. Using the Alert class, the game will show the player a suitable alert if the word is correctly guesses or runs out of attempts, either congratulating them or telling them the game is over.

**Description of 4 features of the game**

The Hangman game successfully implemented 4 core features, each demonstrating the system's functionality and use of object-oriented programming principles. The first feature would be the category selection. The game includes multiple categories such as animals, sports, nature, foods, countries and cars. The categorySelector ComboBox allows the user to switch between categories, updating the word list accordingly. This demonstrates the program's ability to handle different themes and adapt the gameplay.

Next, the second feature is the letter guessing mechanism. Players interact with the game by selecting letters using an on-screen keyboard composed of Button objects. Each time a player guesses a letter, the program checks whether the letter is part of the hidden word (word). Players can guess letters using either an on-screen keyboard or by typing directly into a text field. If the guess is correct, the letter is revealed in the appropriate positions within the guessedWord array. If the guess is incorrect, the number of remaining attempts (attemptsLeft) is decreased. The handleGuess method manages this logic, updating the game state and user interface in real-time. This mechanism allows players to gradually uncover the word while tracking their progress visually.

Furthermore, the system provides visual feedback through hangman images each time an incorrect guesses is made. The hangmanImageView displays a series of images stored in the hangmanImages array, with each image representing a different stage of the hangman figure. As the player makes incorrect guesses, the updateHangmanImage method updates the image, visually showing the player's diminishing chances of guessing the word correctly. This feature not only adds a visual element to the game but also increases the tension as the player sees the hangman figure nearing completion.

Lastly, the last feature is the game state management. The game effectively manages its state, determining when the game is won, lost, or should be reset. If the player successfully guesses all the letters in the word, a congratulatory message is displayed using an Alert, and the keyboard is disabled to prevent further interaction. Conversely, if the player runs out of attempts, the game ends with a "Game Over" message, revealing the correct word. The resetGame method allows the player to start a new game by resetting all game variables, including selecting a new word, resetting the hangman image, and enabling the keyboard. This functionality ensures the game can be replayed without needing to restart the application, offering a seamless user experience.

In summary, the HangmanGame program effectively implements word selection, guessing mechanism, visual feedback through the hangman image, and game state management, providing a fully functional and engaging hangman game that leverages object-oriented programming concepts.

**Personal Reflection**

In developing this Hangman game using Scala and JavaFX, I applied several object-oriented concepts to create a functional and interactive application. The fundamental concepts that I applied are modularity, inheritance, and encapsulation. The main object, HangmanGame, encapsulates the game's state and behavior, demonstrating the principle of encapsulation. This object contains attributes like words, word, and attemptsLeft, which represent the game's current state, and methods like handleGuess and resetGame, which define its behavior. I utilized inheritance by extending the JFXApp class, allowing the HangmanGame object to function as a JavaFX application. This inheritance provides the necessary framework for creating the game's graphical user interface. To make the user interface, JavaFX components such as Label, ImageView, and Button were used in composition. These components are put together inside the HangmanGame object. The implementation also showcases polymorphism, particularly in the event handling for the keyboard buttons. To show how many objects (buttons) might call the same function with varying parameters, the onAction event of each button is set to call the handleGuess method.

During the development of the Hangman game, I encountered challenges such as managing the game state, ensuring UI updates in real-time, and implementing a smooth game reset mechanism. I managed to overcome these challenges by separating the logic into clear methods and making use of ScalaFX's reactive properties to update the UI based on the game's current state. The resetGame method specifically resolved the problem of resuming the game without having to reload the entire program, resulting in a smooth and uninterrupted user experience.

Reflecting on the strengths of my submission, I am confident that the game effectively incorporates the fundamental elements of Hangman, accompanied by an intuitive and user-friendly UI. The code is well-structured, with clear separation of concerns between game logic and UI components. The application of functional programming concepts, such as mapping over the alphabet to generate keyboard buttons, exemplifies a successful combination of object-oriented and functional programming paradigms. However, there are areas for improvement. The current implementation lacks the ability to store high scores or support custom word lists, which could improve the game's replay value. In addition, although the game functions effectively, it would be advantageous to implement more comprehensive error handling and input validation in order to enhance its resilience. The utilisation of object-oriented programming (OOP) characteristics is restricted to encapsulation and modularity, with room for enhancement in terms of error handling to accommodate edge cases and unforeseen user interactions. Overall, this assignment allowed me to deepen my understanding of OOP in a practical context, and while there are areas for improvement, I am satisfied with the outcome and the learning experience it provided.

**References**

Claude. (n.d.). https://claude.ai/

Document. (n.d.). https://howbzr.github.io/portfolio.html