Summative Task: Stock Ticker

In this project, I will be creating a Hangman Game that will be playable for anyone. In this game I will have a bundle of words setup and the player will have to guess the word. You will have 6 failed attempts until you completely fail, giving you the message of the word, or once you have guessed the word it will tell you congratulations. The code will have a hangman character for every time you get a letter wrong and once you get 6 wrong it will be a full character. This code will have a full GUI system with exception handling and it will be a pretty efficient and advanced hangman game. To add on making the decision on picking this game was since I was more familiar with this since I played it as a child.

Requirements:

User Interface:

- A graphical user interface (GUI) to display the game elements.
- Labels to show the word to guess, attempts left, and other relevant information.
- Input fields to allow the user to enter their guesses.
- Buttons to initiate the guessing process and perform other actions.

Game Logic:

- A list of words from which the game will randomly select a word for each session.
- Tracking and updating the guessed letters by the user.
- Counting the remaining attempts and managing game over conditions.
- Implementing the logic to reveal correct guesses in the word.

Hangman Art:

- ASCII art or graphical representation of the hangman character.
- Different stages or versions of the hangman character for each incorrect guess.

User Interaction:

- Capturing user input and validating it (single letter, non-repetitive guesses, etc.).
- Displaying appropriate messages to the user based on their input and game progress.
- Providing feedback on the game outcome (win or lose).

Error Handling:

- Handling unexpected input or errors gracefully.
- Ensuring the program does not crash or become unresponsive due to incorrect user input.

Documentation and Instructions:

- Providing clear instructions on how to play the game.

- Adding comments and documentation within the code to enhance understanding.

Testing and Debugging:

- Conducting thorough testing of the program to identify and fix any issues or bugs.
- Ensuring the game functions as expected in various scenarios.

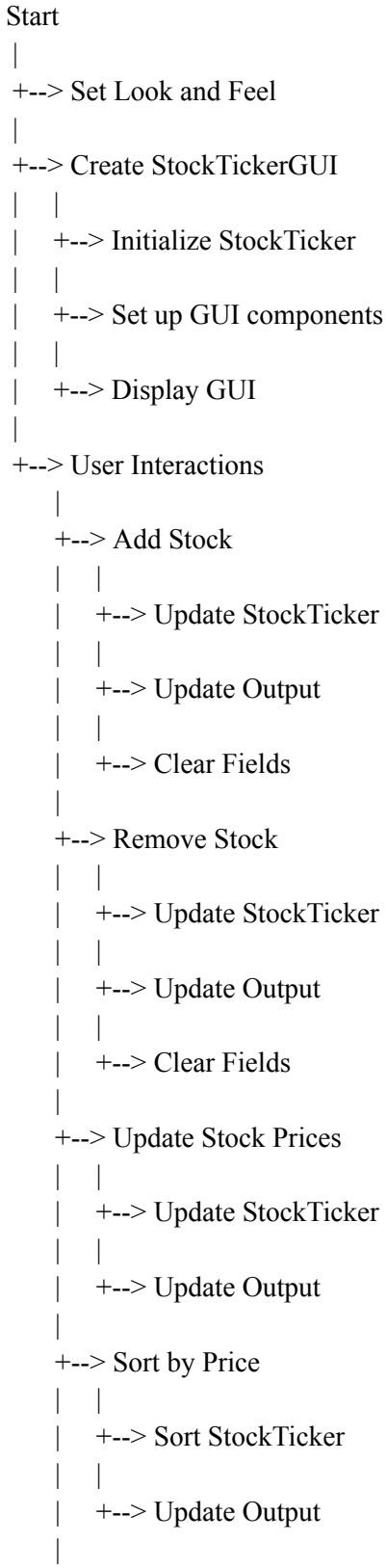
Code Organization and Structure:

- Writing clean, modular, and well-organized code.
- Separating concerns into different functions or classes.
- Using appropriate naming conventions and coding best practices.

General Algorithm:

1. Create a word list containing a collection of words for the game.
2. Initialize the GUI components, including labels, input fields, buttons, and hangman character display.
3. Set the initial game state by selecting a random word from the word list.
4. Display the masked word, attempts left, and other relevant information on the GUI.
5. Handle user input:
    a. Listen for the "Guess" button click event.
    b. Retrieve the input from the input field.
    c. Validate the input (single letter, non-repetitive guess).
    d. Update the game state based on the user's guess:
    e. Check if the guessed letter is present in the word.
    f. Update the masked word and guess the letters accordingly.
    g. Decrement the attempts left if the guess is incorrect.
    h. Update the hangman character display based on the attempts left.
    i. Check for game over conditions:
    j. If the player has no attempts left, display a losing message and end the game.
    k. If the masked word has no remaining underscores, display a winning message and end the game.
    l. Update the GUI to reflect the changes in the game state.
    m. Clear the input field and set the focus back to it for the next guess..
6. Handle the end of the game:
    a. Disable the input field and "Guess" button.
    b. Display an appropriate end game message
7. Provide an option to start a new game:
    a. Listen for a new game event or button click.
    b. Reset the game state, including attempts left, guessed letters, and hangman character.
    c. Select a new random word from the word list.
    d. Enable the input field and "Guess" button.
    e. Update the GUI to reflect the new game state.
8. Implement any additional features, such as score tracking or game statistics, as desired.
9. Test the program thoroughly, considering different scenarios and edge cases.
10. Refactor and optimize the code as needed for better readability and performance.

11. Document the code and provide clear instructions for users on how to play the game.



+--> Save to File

|

Prototype:

1) Word to Guess: The center panel displays the word to guess with underscores representing unguessed letters. Initially, all letters are hidden.

2) Attempts Left: The panel shows the number of attempts left for the player to guess the word correctly.

3) Input Field: A text field where the user can enter their guesses for the word. Only single letters are allowed as valid guesses.

4) Guess Button: Clicking the "Guess" button submits the user's guess. The program validates the guess and updates the game state accordingly.

5) Hangman ASCII Art: A text area that displays ASCII art representing the hangman. The art is updated based on the number of attempts left.

6) End of Game: When the game is over, the program displays a message dialog showing whether the player has won or lost. The "Guess" button and input field are disabled.

7) Restart Game: After the game ends, the player can click the "Restart Game" button to start a new game. The word to guess, attempts left, and hangman ASCII art are reset..

8) Exit: The player can exit the game by closing the window or clicking an "Exit" button, if available.

==========
Welcome to Hangman Game!
==========

Attempts left: 6

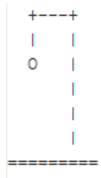
Word to guess: _ _ _ _ _ _

Enter your guess: a

Attempts left: 6

Word to guess: _ a _ _ a _

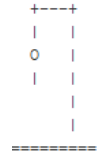
Enter your guess: e



Attempts left: 5

Word to guess: _ a _ _ a _

Enter your guess: i

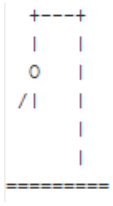


Attempts left: 4

Word to guess: _ a _ i a _

Enter your guess: o

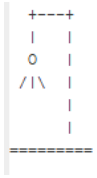
Attempts left: 3



Word to guess: _ a _ i a _

Enter your guess: u

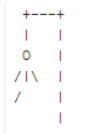
Attempts left: 2



Word to guess: _ a u i a _

Enter your guess: t

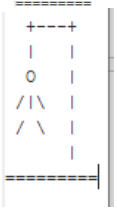
Attempts left: 1



Word to guess: _ a u i a _

Enter your guess: p

You lose! The word was: hangman



Pseudocode 1:

Create a HangmanGame class
    Initialize wordList with a list of words
    Initialize wordToGuess as an empty string
    Initialize guessedLetters as an empty set
    Initialize attemptsLeft to 6

    Function getRandomWord():
        Select a random word from wordList
        Return the randomly selected word

    Function initializeGame():
        wordToGuess = getRandomWord()
        Display welcome message
        Display the initial state of the hangman

    Call the initializeGame() function

Pseudocode 2:

```
Function processGuess(guess):
    If guess is a single letter:
        If guess is already in guessedLetters:
            Display an error message ("You already guessed that letter.")
        Else:
            Add guess to guessedLetters
            If guess is not in wordToGuess:
                Decrease attemptsLeft by 1
    Else:
        Display an error message ("Please enter a single letter.")

    Call updateUI() to update the game state
```

Pseudocode 3:

```
Function updateUI():
    Clear the previous output
    Display the current state of the hangman
    Display the masked word with correctly guessed letters
    Display the number of attempts left

    If the game is won:
        Display a winning message
        Display the actual word
        Prompt for starting a new game
    Else if the game is lost:
        Display a losing message
        Display the actual word
        Prompt for starting a new game
```

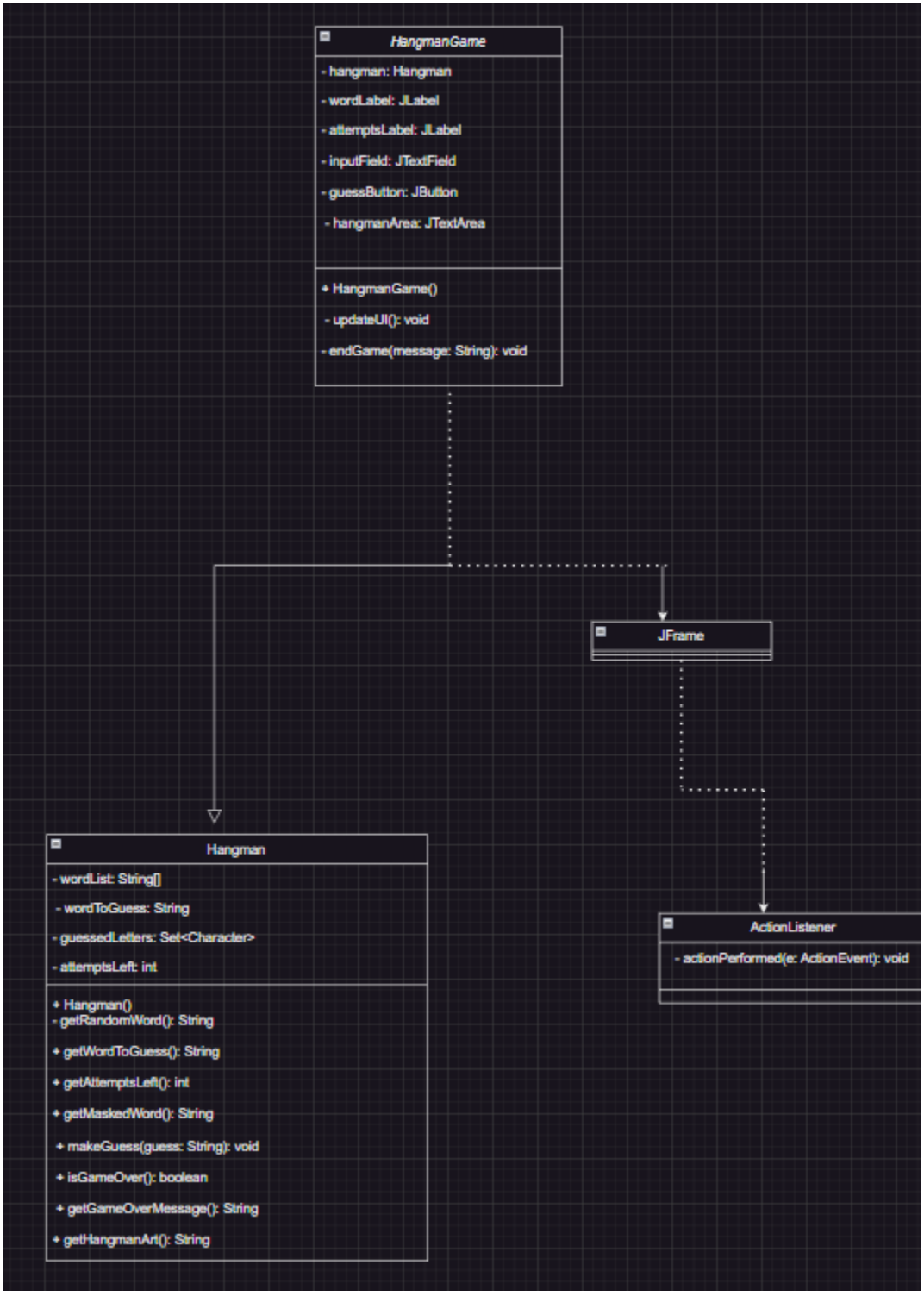
Pseudocode 4:

```
Function displayHangman():
    Create an array of hangman art for each stage
    Display the hangman art based on attemptsLeft

Function clearOutput():
    Clear the output area

Function displayMessage(message):
    Display the provided message in the output area
```

UML Diagram:


HangmanGame
- hangman: Hangman
- wordLabel: JLabel
- attemptsLabel: JLabel
- inputField: JTextField
- guessButton: JButton
- hangmanArea: JTextArea
+ HangmanGame()
- updateUI(): void
- endGame(message: String): void
JFrame
Hangman
- wordList: String[]
- wordToGuess: String
- guessedLetters: Set<Character>
- attemptsLeft: int
+ Hangman()
- getRandomWord(): String
+ getWordToGuess(): String
+ getAttemptsLeft(): int
+ getMaskedWord(): String
+ makeGuess(guess: String): void
+ isGameOver(): boolean
+ getGameOverMessage(): String
+ getHangmanArt(): String
ActionListener
- actionPerformed(e: ActionEvent): void

Test and bug:

-Testing with variety of Inputs

Test Case	Action	Result
1	User enters “Z”	Lost a point
2	User enters”z"	Correct
3	User enters”zebra”	The first letter is taken as a guess

Test Case	Action	Result
1	User enters “1456”	Lost a point
2	User enters”**##$"	Lost a point
3	User enters””	Lost a point

-Feedback

Errors	Action
Game treats the uppercase and lowercase version of a correct letter differently.	Added toLowerCase() method on top of getText() to convert the guessField input to lowercase.
GUI System functionality was inefficient	Used ASCII value instead for Stickman

Hangman Daily Calendar

DATE	TASK ACCOMPLISHED	Log
Apr 19, 2023	Selected Idea for project	✔
Apr 23, 2023	Begin planning using SDLC.	✔
Apr 24, 2023	Continue Planning using SDLC.	✔
Apr 25, 2023	Finish Start of Plan and Start Algorithm	✔
Apr 26, 2023	Finish Algorithm	✔
Apr 27, 2023	Start Prototype	✔
Apr 28, 2023	Finish Prototype and Start Pseudocode	✔
Apr 30, 2023	Finish Pseudocode and Begin Code	✔
May 1, 2023	Continue Code	✔
May 3, 2023	Continue Code	✔
May 5, 2023	Final Review	✔
May 8, 2023	Present Project	✔