UNIVERSITY OF ST ANDREWS

HANGMAN

230010723

MSc Computer Science

Software Engineering Practice

12, FEBRUARY 2024

Contents

0.1 Overview (How to Play)

Hangman is a word-guessing game where one player thinks of a word and the other player tries to guess it by suggesting letters within a certain number of attempts. In this implementation, there are three main components:

0.1.1 Hangman

The Hangman class acts as the entry point to the Hangman game. It contains the main method, which initializes the HangmanController and HangmanView instances, sets up the game, and starts the game loop.

0.1.2 HangmanView

This class handles the user interface of the game. It displays the Hangman ASCII art representing the current state of the game (number of incorrect guesses), prompts the user for input, and displays feedback such as the number of chances left and the incorrect letters guessed.

0.1.3 HangmanController

This class manages the game logic. It selects a random word from a predefined list, initializes the game settings such as the number of chances allowed, and tracks the correct and incorrect guesses. It also handles user input validation and updates the hidden word to reveal correct guesses. Additionally, it determines whether the game has ended, whether the player won or lost.

0.2 Test-Driven Approach

Before writing any code, I began by identifying various scenarios to test the Hangman game thoroughly. These scenarios covered critical aspects such as word selection, game initialization, user input validation, and game ending conditions. Once I had a clear understanding of the test scenarios, I proceeded to create corresponding test cases using JUnit. Each test case was designed to address a specific scenario, ensuring comprehensive test coverage.

Following the creation of test cases, I executed the initial set of tests to verify their effectiveness. It was essential to ensure that all tests failed initially, indicating that they were indeed testing the intended behavior and not mistakenly passing. With failing test cases in place, I began implementing the game logic incrementally. For each test case, I wrote the minimum amount of code required to make it pass, adhering to the test-driven development (TDD) approach.

After successfully implementing the logic to make a test case pass, I performed a thorough code review and refactored the code as necessary. Refactoring aimed to improve readability, maintainability, and adherence to best practices. This iterative

process continued until all test cases passed, signifying the correctness and robustness of the Hangman game implementation.

0.3 Refactoring

Throughout the development process, I adopted a proactive approach to refactoring, prioritizing the maintenance of clean, readable, and maintainable code. Refactoring wasn't just an afterthought; it was integrated as a crucial aspect of the development life cycle. I continuously sought opportunities for improvement, aiming for a code base that was not only functional but also elegant and easy to understand.

Identifying areas for improvement was a key aspect of the refactoring process. I paid close attention to common code smells such as duplication, lengthy methods, ambiguous variable names, and convoluted conditional logic. These served as signals for potential areas that could benefit from refactoring. Instead of attempting large-scale changes all at once, I opted for small, incremental adjustments to the codebase. This approach minimized the risk of introducing new bugs or unintended side effects while gradually improving the overall quality of the code.

Leveraging the powerful refactoring tools provided by my Integrated Development Environment (IDE) was instrumental in streamlining the refactoring process. IntelliJ IDEA offered a range of features to facilitate common refactoring tasks, allowing me to make changes efficiently and with confidence

0.4 Other Software Development Practices Used

Modular Design played a pivotal role in structuring the Hangman game code base. By breaking down the implementation into modular components, I aimed to facilitate code reuse and maintainability. Each module focused on a specific aspect of the game, such as game logic, user interface, or input/output handling, thereby promoting clarity and organization within the code base.

Adhering to SOLID principles guided the design and implementation of the Hangman game components. I aimed to create cohesive and loosely coupled components with clearly defined responsibilities and interfaces. This approach not only facilitated easier code maintenance but also promoted flexibility and extensibility in the face of future changes or enhancements.

Version Control using Git and GitHub served as the backbone of the development process, enabling collaborative development and efficient code management. By leveraging Git for version control and GitHub for hosting the repository, I established a centralized platform for collaboration, code review, and version tracking. It ensured code integrity and version traceability.

0.5 Conclusion

The Hangman game implementation followed a systematic approach, incorporating TDD, refactoring, and other software development practices to ensure high code quality, reliability, and maintainability. By adopting these practices, the Hangman game provides an enjoyable and interactive experience for players.