**Matchle Program Design Improvement Document**

https://github.com/PercyQian/Matcher.git

**I. Project Overview**

Matchle is a word-guessing game application available in both console and GUI versions. This document outlines the improvements made to the original design with a focus on clearer functional division, optimized code structure, enhanced maintainability, and robust error handling.

**II. Design Improvement Approach**

**A. Main Optimization Principles**

* **Separation of Concerns:**  
  Game logic is strictly separated from user interface code.
* **Single Responsibility:**  
  Each class is designed to handle only one specific functionality.
* **Complexity Reduction:**  
  Complex methods have been split into smaller, more manageable components.
* **Code Organization:**  
  Clear comments and logical grouping are used throughout the code to enhance readability.
* **Redundancy Elimination:**  
  Unnecessary code and duplicate functionality have been removed.

**B. Core Architecture Improvements**

The improved architecture is illustrated by the following class structure diagram:

**III. Class Design Details**

**1. GameLogic Class**

**Responsibility:**  
Encapsulates all core game logic, independent of the presentation layer.

**Key Attributes:**

* Corpus originalCorpus: The original word corpus.
* NGram secretKey: The secret keyword selected for the game.
* Corpus candidateCorpus: The current candidate words based on game progress.
* Filter accumulatedFilter: The cumulative filter applied after each guess.

**Main Methods:**

| **Method** | **Parameters** | **Return Type** | **Preconditions** | **Postconditions** |
| --- | --- | --- | --- | --- |
| initialize | Corpus corpus, NGram secretKey | void | Corpus is not null | Game state is set using provided corpus/key |
| initialize | Corpus corpus | void | Corpus is not null | Game state is initialized with a random key |
| createDefaultCorpus | *none* | Corpus | *none* | Returns a default corpus |
| isCorrectGuess | NGram guess | boolean | Guess is not null | Returns whether the guess is correct |
| processGuess | NGram guess | Filter | Guess is not null | Updates game state; returns the round filter |
| getBestGuess | *none* | NGram | Candidate corpus exists | Returns the best guess word |
| hasGameTerminated | *none* | boolean | *none* | Indicates whether the game has ended |
| getSecretKey | *none* | NGram | *none* | Returns the secret keyword |
| getCandidateCorpus | *none* | Corpus | *none* | Returns the filtered candidate corpus |

**Algorithm Overview:**

* **processGuess:**  
  Generates a filter based on the user’s guess compared against the secret keyword. It updates the accumulated filter and reduces the candidate corpus by eliminating non-matching words.
* **updateCandidateCorpus:**  
  Applies the updated filter to the candidate corpus to remove any words that do not meet the game’s conditions.

**2. MatchleGame Class**

**Responsibility:**  
Manages the command-line interface and console interactions.

**Key Attributes:**

* GameLogic gameLogic: The game engine.
* Corpus corpus: The word corpus used in the game.
* int maxRounds: The maximum number of rounds allowed.

**Main Methods:**

| **Method** | **Parameters** | **Return Type** | **Preconditions** | **Postconditions** |
| --- | --- | --- | --- | --- |
| initialize | *none* | void | *none* | Game is initialized |
| loadCorpus | *none* | void | *none* | Corpus is successfully loaded |
| play | *none* | void | Game has been initialized | Game execution completes |
| playRound | *none* | boolean | *none* | Executes a round and returns a game-over flag |
| makeGuess | *none* | NGram | *none* | Returns the guess provided for the round |
| checkGameTermination | *none* | boolean | *none* | Determines if the game should terminate |

**Improvements:**

* Grouped methods with clear comment blocks for readability.
* Simplified the play method by reducing unnecessary parameter passing.
* Enhanced method documentation to provide detailed usage information.

**3. MatchleGUI Class**

**Responsibility:**  
Provides a graphical user interface and manages user interactions through various UI components.

**Key Attributes:**

* GameLogic gameLogic: The underlying game logic engine.
* Corpus corpus: The word corpus utilized in the game.
* Various UI components such as JTextField guessField, JTextArea feedbackArea, etc.
* Theme-related attributes including dedicated color constants and state variables for dark/light mode.

**Main Methods:**

| **Method** | **Parameters** | **Return Type** | **Preconditions** | **Postconditions** |
| --- | --- | --- | --- | --- |
| initUI | *none* | void | *none* | UI is fully initialized |
| startNewGame | *none* | void | UI must be initialized | A new game session is started |
| submitGuess | *none* | void | *none* | Processes the user’s guess |
| updateGameState | NGram guess | void | Guess is not null | Updates and displays the new game state |
| handleGameState | boolean isSaving | void | *none* | Saves or loads game state based on flag |
| toggleDarkMode | *none* | void | *none* | Switches between dark and light themes |

**Improvements:**

* UI initialization is divided into multiple methods to minimize complexity.
* Event handling has been optimized by incorporating lambda expressions.
* Unused methods have been removed.
* Enhanced theme switching by adopting more efficient algorithms (such as using breadth-first approaches instead of recursive methods).

**4. Support Class Enhancements**

**4.1 Filter Class**

**Improvements:**

* The unused FALSE constant has been removed; filters are now created directly when needed.
* A factory method pattern has been standardized, with the constructor made private to control object creation.

**4.2 CorpusLoader Class**

**Improvements:**

* Removed test code (such as the main and testHardCase methods) as they are no longer needed in the production environment.
* Added input validation to ensure that invalid or null inputs do not cause unexpected behaviors.

**4.3 Exception Handling**

**Improvements:**

* Standardized the management of serialVersionUID by defining it only in the top-level exception class.
* Implemented a custom exception framework with specific exception types (e.g., EmptyCorpusException) to improve error clarity and recovery.

**IV. Error Handling and Defensive Programming**

**A. Input Validation**

* **User Input:**  
  The MatchleGUI.validateAndGetGuess() method now includes rigorous checks.
* **Game Logic:**  
  Null checks have been added to critical methods, such as in GameLogic.processGuess().
* **Corpus Loading:**  
  The CorpusLoader.loadEnglishWords() method now validates input results to prevent null returns.

**B. Exception Handling**

* Exception handling is performed at the UI layer using try-catch blocks to gracefully manage any game logic errors.
* A unified error message display mechanism has been implemented to provide users with detailed error information and recovery suggestions.

**C. Defensive Programming Practices**

* Key methods now incorporate parameter validations to ensure stability.
* State variables are strictly initialized before use.
* Measures have been taken to prevent null reference issues that could lead to system crashes.

**V. Testing Strategy**

**A. Unit Tests**

**GameLogic Class Tests:**

| **Test Method** | **Purpose** | **Verification Points** |
| --- | --- | --- |
| testInitialize | Validate game initialization | Confirm that the game state is set properly |
| testProcessGuess | Evaluate guess processing | Ensure filters are generated and state updates occur |
| testIsCorrectGuess | Verify correctness of guesses | Check if the method accurately determines the guess |

**MatchleGame Class Tests:**

| **Test Method** | **Purpose** | **Verification Points** |
| --- | --- | --- |
| testPlayRound | Test single round mechanics | Validate game state updates and termination checks |
| testCheckGameTermination | Assess game termination logic | Ensure correct evaluation of game termination conditions |

**Filter Class Tests:**

| **Test Method** | **Purpose** | **Verification Points** |
| --- | --- | --- |
| testFilterCreation | Validate correct filter creation | Confirm that filters are created using factory methods |
| testFilterAnd | Evaluate filter combination | Ensure that logical AND operations on filters work as expected |

**B. Private Method Test Hooks**

To facilitate testing of private methods, a test hook class is provided. For example:

// Test hook example

class GameLogicTestHook extends GameLogic {

public Filter publicGenerateRoundFilter(NGram guess) {

return super.generateRoundFilter(guess);

}

public void publicUpdateCandidateCorpus() {

super.updateCandidateCorpus();

}

}

**C. Stress Test Plan**

**Objective:**  
Assess system performance and stability using a large corpus and extended runtime scenarios.

**Test Description:**

1. Load a large corpus containing approximately 10,000 words.
2. Execute 100 continuous game rounds with random target word selection for each round.
3. Simulate player input by processing at least 10 guesses per round.
4. Record the runtime and memory usage for each game round.
5. Verify that there are no memory leaks or significant performance degradations.

**Success Criteria:**

* All game rounds complete successfully.
* Average response time is under 200 milliseconds.
* Memory usage remains stable without continual growth.
* The random word selection function demonstrates a uniform distribution.

**VI. Conclusion**

The improvements detailed in this document optimize the Matchle program by restructuring the code, separating game logic from the user interface, and introducing modular design practices. Enhanced error handling, thorough input validation, and comprehensive testing strategies work together to ensure robustness, efficiency, and maintainability. These changes not only streamline future development and maintenance but also significantly improve overall program stability and user experience.

classDiagram

class GameLogic

class MatchleGame

class MatchleGUI

class GameState

class GameStateManager

class Corpus

class NGram

class Filter

class CorpusLoader

MatchleGame --> GameLogic : uses

MatchleGUI --> GameLogic : uses

GameLogic --> GameState : creates/loads

GameLogic --> Corpus : maintains

GameLogic --> NGram : maintains

GameLogic --> Filter : maintains

MatchleGUI --> GameStateManager : uses

GameStateManager --> GameState : saves/loads

Corpus --> NGram : contains

GameState --> NGram : contains

GameState --> Corpus : contains

GameState --> Filter : contains

Filter --> NGram : filters