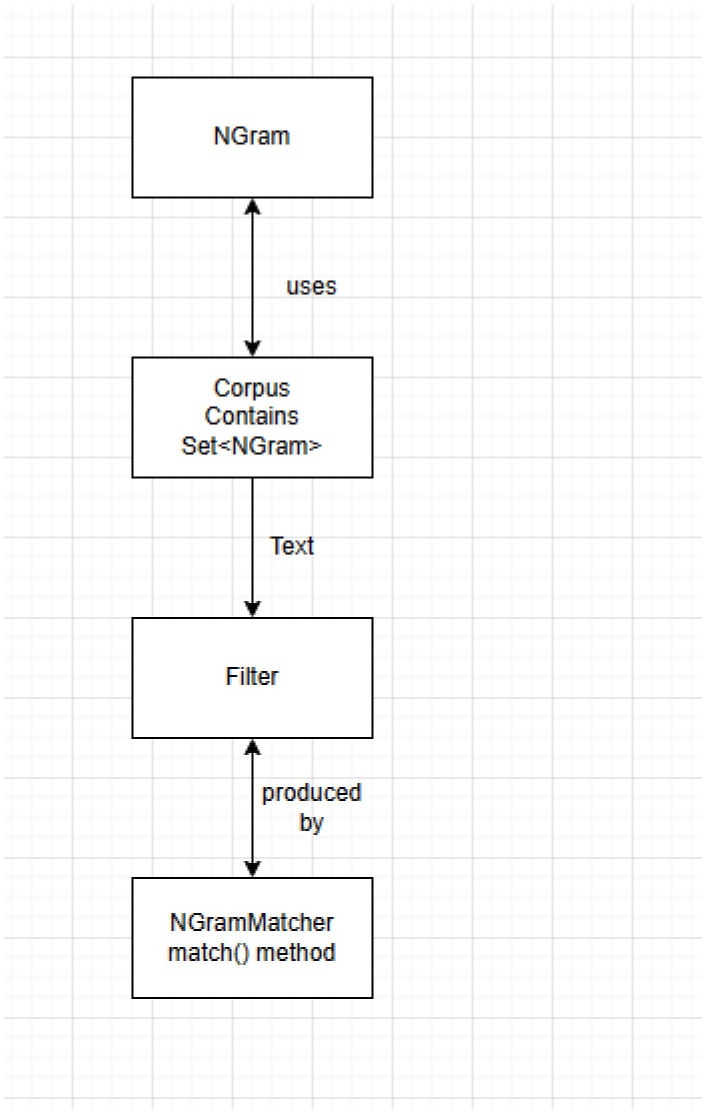
General Idea

The system is designed as a modular, object-oriented Java project.

- NGram is an immutable value object representing a word.
- Corpus is a container of valid NGram words; its internal collection is immutable.
- Filter encapsulates the feedback rule (similar to Wordle's green/yellow/gray) as a Predicate over NGrams.
- NGramMatcher implements a three-pass matching algorithm, returning a Filter that specifies which letters in a guess are exact, misplaced, or absent.
- MatchleScorer uses the matcher and corpus to score guesses (worst-case and average-case) and select optimal guesses.
- 

2. Detailed Class Descriptions

2.1 NGram

- Attributes:
    - private final ArrayList<Character> ngram;
      Stores the characters of the word in order.
    - private final Set<Character> charset;
      A helper set containing unique letters (for quick membership testing).
- Routines:
    - static NGram from(String str): NGram
      Input: A non-null String.
      Precondition: str ≠ null (throws NullPointerException if violated).
      Postcondition: Returns a new immutable NGram with size equal to string length.
    - static NGram from(List<Character> chars): NGram
      Input: A non-null List of Characters containing no null elements.
      Precondition: List and all elements are non-null.
      Postcondition: Returns a new NGram instance.
    - Character get(int index): Character
      Input: An int index.
      Precondition: 0 ≤ index < ngram.size() (throws IndexOutOfBoundsException if violated).
      Postcondition: Returns the character at the given index.
    - int size(): int
      Returns the number of characters in this NGram.
    - boolean matches(IndexedCharacter c): boolean
      Input: An IndexedCharacter (record with index and character).
      Precondition: c ≠ null.
      Postcondition: Returns true if the character at c.index equals c.character.
    - boolean contains(char c): boolean
      Returns true if the character c is contained in the NGram.
    - boolean containsElsewhere(IndexedCharacter c): boolean
      Input: An IndexedCharacter.
      Precondition: c ≠ null.

Postcondition: Returns true if c.character is present in a different index than c.index.

  - Iterator and Stream methods:
    Provide iteration over IndexedCharacter objects.

- Error Handling:
  - All public routines check for null inputs and enforce index bounds.
  - Defensive copying is used in factory methods to preserve immutability.
- Testing:
  - Unit tests should verify correct construction, equality, immutability, iterator behavior, exception cases (null/invalid index), and correct responses from methods like matches, contains, containsElsewhere.

2.2 Corpus

- Attributes:
  - private final Set<NGram> corpus;
    An unmodifiable set of NGram objects (built by copying the input set).
  - private final int wordSize;
    All words in the corpus have this common length.
- Routines:
  - Set<NGram> corpus(): Set<NGram>
    Returns the unmodifiable set of NGrams.
  - int wordSize(): int
    Returns the common word length.
  - int size(): int
    Returns the number of NGrams in the corpus.
  - boolean contains(NGram ngram): boolean
    Input: Non-null NGram.
    Precondition: ngram ≠ null.
    Returns whether the corpus contains the NGram.
  - Iterator<NGram> iterator() and Stream<NGram> stream()
    Return iterators or streams over the corpus.

    - long size(Filter filter): long
      Input: A non-null Filter.
      Returns the count of NGrams in this corpus satisfying the filter.

- Nested Builder Class:
    - Attributes:
        - private final Set<NGram> ngrams;
          A modifiable set used to accumulate NGrams before building the final Corpus.
    - Routines:
        - Builder add(NGram ngram): Builder
          Input: Non-null NGram; adds it to the builder.
        - Builder addAll(Collection<NGram> collection): Builder
          Adds all non-null NGrams from the collection.
        - boolean isConsistent(Integer wordSize): boolean
          Input: Non-null Integer.
          Returns true if all NGrams in the builder have the given wordSize.
        - Corpus build(): Corpus
          Precondition: All NGrams are non-null and of equal length.
          Postcondition: Returns an immutable Corpus or throws IllegalStateException if inconsistent.
- Error Handling:
    - All public methods check for null.
    - The Builder's build() method enforces consistent state.
- Testing:
    - Unit tests should create corpora using the Builder.
    - Test immutability (try to modify the corpus, expect an exception).
    - Verify proper behavior when null values are added or inconsistent sizes appear.

2.3 Filter

- Attributes:

- private final Predicate<NGram> predicate;
  The internal condition that a candidate NGram must meet.

- Routines:
  - static Filter from(Predicate<NGram> predicate): Filter
    Input: A non-null Predicate.
    Precondition: predicate ≠ null.
    Returns: A Filter encapsulating the predicate.
  - boolean test(NGram ngram): boolean
    Input: A non-null NGram.
    Returns: The result of applying the predicate.
  - Filter and(Optional<Filter> other): Filter and Filter and(Filter other): Filter
    Combine this Filter with another using logical AND.
- Error Handling:
  - Every method validates its input.
  - The internal predicate is set only if not null.
- Testing:
  - Unit tests should check that filters behave as intended (true/false for given NGrams), including combining filters (and) and handling null parameters (which should throw exceptions).

2.4 NGramMatcher

- Attributes:
  - private final NGram key; – The secret word to be matched.
  - private final NGram guess; – The player's guess.
- Routines:
  - static NGramMatcher of(NGram key, NGram guess): NGramMatcher
    Input: Two non-null NGrams.
    Precondition: key and guess ≠ null.
    Returns: A new NGramMatcher instance.
  - Filter match(): Filter
    Returns: A Filter representing the feedback comparing key and guess. *Algorithm:* (See pseudocode below)

        - *Precondition:* key.size() == guess.size(); if not, returns Filter.FALSE.
        - *Postcondition:* The returned Filter combines conditions on each letter position:
            - Exact Matches (Green): For every i where key[i] equals guess[i], add condition that candidate must have that letter at position i.
            - Misplaced Matches (Yellow): For each unmatched guess letter, search for a matching letter in an unmatched position of key; add condition that candidate should contain the letter elsewhere.
            - Absent (Gray): For guess letters that remain unmatched, add condition that candidate must not contain that letter.
    - Helper Methods (private):
        - doExactMatches(...)
          For each index i, if key[i] equals guess[i], mark as matched and add an exact match Filter.
        - doMisplacedMatches(...)
          For each index not matched, call handleMisplacedAtIndex(...) *to check for misplaced letters, and add a misplaced Filter.*
        - doAbsentCharacters(...)
          For each unmatched position, add an absent Filter.
        - handleMisplacedAtIndex(int i, int n, boolean[] keyMatched, boolean[] guessMatched, List<Filter> partialFilters)
          Processes a single guess index i to find a match in any unmatched position j in key; if found, marks them as matched and adds a misplaced filter.
- Error Handling:
    - The factory method uses Objects.requireNonNull for key and guess.
    - Early exit in match() when lengths differ.
- Testing:
    - Unit tests should cover:
        - Different lengths (expect Filter.FALSE).
        - Exact match only.
        - Misplaced match (with duplicate letters).

        - Absent letters.
    - Private helper methods can be indirectly tested via the public match() method.
    - Test invalid cases (null inputs; out-of-bound indices through NGram methods)

Pseudocode Summary for NGramMatcher.match():

```
function match(guess, target) -> Filter:
  if guess.length != target.length:
    return Filter.FALSE

  initialize keyMatched[ ] and guessMatched[ ] of size n to false
  initialize partialFilters as empty list

  // PASS 1: Exact matches
  for each index i in 0 to n-1:
    if guess[i] == target[i]:
      mark keyMatched[i] and guessMatched[i] true
      add exact match filter at i to partialFilters

  // PASS 2: Misplaced matches
  for each index i in 0 to n-1:
    if guessMatched[i] is false:
      call handleMisplacedAtIndex(i)
        for each j in 0 to n-1:
          if keyMatched[j] is false and target[j] equals guess[i]:
            mark keyMatched[j] and guessMatched[i] true
            add misplaced filter for i to partialFilters
            break
```

```
// PASS 3: Absent letters
for each index i in 0 to n-1:
    if guessMatched[i] is false:
        add absent filter for guess[i] to partialFilters

Combine all partialFilters using logical AND
return the final Filter
```

2.5 MatchleScorer

- Attributes:
  - private final Corpus corpus; – The immutable dictionary of NGrams.
- Routines:
  - MatchleScorer(Corpus corpus): void
    Input: A non-null, non-empty Corpus.
    Precondition: corpus ≠ null and corpus.size() > 0.
    Postcondition: Stores the corpus in an immutable field.
  - long score(NGram key, NGram guess): long
    Inputs: Non-null key and guess.
    Precondition: key and guess are not null.
    Process: Uses NGramMatcher.of(key, guess).match() to generate a Filter.
    Return: The count of NGrams in the corpus that satisfy the feedback filter.
  - long scoreWorstCase(NGram guess): long
    Input: A non-null guess.
    Return: The maximum score over all keys in the corpus (worst-case scenario).
  - long scoreAverageCase(NGram guess): long
    Input: A non-null guess.
    Return: The sum of scores over all keys in the corpus.
  - NGram bestGuess(ToLongFunction<NGram> criterion): NGram
    Input: A non-null criterion function mapping a candidate guess to a long score.
    Return: The NGram in the corpus with the lowest score by the criterion.
  - NGram bestWorstCaseGuess(): NGram
    Delegates: Calls bestGuess(this::scoreWorstCase).

    - NGram bestAverageCaseGuess(): NGram
      Delegates: Calls bestGuess(this::scoreAverageCase).
- Error Handling:
    - All methods use defensive checks (Objects.requireNonNull) for inputs.
    - The constructor throws an exception if corpus is null or empty.
- Testing:
    - Unit tests for all scoring methods verifying expected outputs.
    - Tests verifying that passing null causes NullPointerException.

---

3. Error Handling and Defensive Strategies

- Defensive Checks:
    - Every public method uses Objects.requireNonNull(...) to check for nulls.
    - Preconditions on methods (such as matching lengths in NGramMatcher.match) are checked early; if violated, a known (false) value or exception is returned.
- Immutability:
    - NGram and Corpus objects are immutable.
    - Defensive copies are made in constructors (e.g., Corpus and NGram).
- Clear Exceptions:
    - Methods throw specific exceptions (NullPointerException, IllegalStateException, IndexOutOfBoundsException) with descriptive messages.

---

4. Testing Strategy

- Unit Tests:
    - Write a dedicated test class for each core component:
        - NGramTest: Verify construction (from String/List), correct element access, iteration, equals/hashCode, boundary conditions.
        - CorpusTest: Verify that the Builder correctly accumulates NGrams, enforces consistency, and that public methods return the expected unmodifiable view.

        - FilterTest: Verify that the Filter factory methods, test(), and the logical AND methods behave as expected and enforce null checks.
        - NGramMatcherTest: Verify that match() correctly computes feedback (exact, misplaced, absent) for various key-guess pairs.
        - MatchleScorerTest: Verify scoring methods (score, worst-case, average-case, best guess methods) return expected values; also test defensive behavior against null inputs.
- Integration Tests:
    - Simulate a game loop where a series of guesses narrow down the Corpus until the secret is found.
    - Ensure that filtering and scoring are consistent across rounds.
- Stress Test:
    - Create a large Corpus (e.g., using thousands of 5-letter words from an extended dictionary).
    - Run through many simulated games in a loop, ensuring that:
        - Performance remains acceptable.
        - No memory leaks occur (by monitoring memory usage over iterations).
        - All defensive conditions remain true, and no invalid state is reached.
    - This single stress test would run a loop (perhaps tens of thousands of iterations) that randomly selects a secret, simulates guesses using the scoring functions, and verifies that the system consistently converges on a valid answer.
- Test Hooks:
    - For private methods (especially in NGramMatcher), design your tests to cover them indirectly via the public match() method.
    - If needed, use reflection (or package-private visibility) to test edge cases for private methods, though ideally the public behavior covers the important logic.

---

5. Single Stress Test Description

Stress Test Outline:

- Purpose: Test the performance and stability of the scoring and matching system with a large corpus and many iterations.

- Setup:
  - Load a large dictionary of NGrams (e.g., 10,000 words of a fixed length).
  - Build a Corpus from this dictionary.
- Test Process:
  - For a fixed number of iterations (e.g., 10,000 rounds), randomly select a secret key from the Corpus.
  - For each iteration, simulate a guess sequence:
    1. Use MatchleScorer to compute worst-case or average-case scores for a set of candidate guesses.
    2. Select the best guess.
    3. Use NGramMatcher.match() to generate a feedback Filter.
    4. Filter the Corpus and update candidate list.
  - Verify that after a sequence of rounds (or simulation), the candidate set is reduced to one word—the chosen secret.
  - Record timing and monitor memory usage.
- Validation:
  - Ensure each iteration completes without errors or exceptions.
  - Confirm that the system's performance scales linearly (or better) with the number of words.
  - The stress test should run unattended, logging summary metrics (total runtime, average time per round, memory usage).