CSDS 293
Software Craftsmanship
2025 Spring Semester

Programming Assignment 4

Due at the beginning of your discussion session on

February 10-14, 2025

Reading

In addition to the following topics, the quiz syllabus includes any material covered in the lectures:

- In chapter 5 of Code Complete:
  - Section 5.1
  - Figure 5-2
  - "Form Consistent Abstractions"
  - "Encapsulate Implementation Details"
  - "Hide Secrets (Information Hiding)"
  - "Keep Coupling Loose"
- Section 19.6 in Code Complete
- Item 64 in Effective Java

Special Grading Guidelines

Starting with this assignment, an automatic C (or less) is triggered by:

- Any routine with complexity greater than 4, or by
- Any piece of code that is essentially repeated.

However, a submission that avoids these problems does not necessarily qualify for high quality craftsmanship.

Programming

First, make all the changes discussed in your discussion section. Additionally, you should refactor your code to make sure that it adopts the principles covered in the reading assignments.

Scoring

In the game of matchle, at each round, the player formulates a guess. The ultimate objective is to use as few guesses as possible to discover the hidden key. To create a guess, the player will examine all available n-grams and choose one that reduces the number of keys as much as possible. Indeed, if she succeeds in reducing the number of possible keys to one, then she is done, otherwise she can continue with a potentially much smaller corpus of candidate keys. In other words, the player should submit a guess that reduces the corpus' size as much as possible. In `Corpus`, define

```
public long score(NGram key, NGram guess)
```

as the size of the corpus that is consistent with the filter that matches the key and the guess n-grams. The idea is that once the player receives information about the congruency of key and guess, the filter can be used to reduce the corpus size. The ideal guess should the one that minimizes the score. The `score` method, and all its derivatives below, throw an `IllegalStateException` if the `Corpus` is empty, preventing the calculation of the `score`.

However, the score function is unusable in this form because it assumes that the player already knows the key, which is exactly what she is trying to find. To remediate the problem, the player will run through all corpus' key and make a guess regarding the n-gram with the maximum score. The idea is that, since the player does not know the key, she would guess a n-gram that results in the smallest corpus size when the key is the worst possible one. In `Corpus`, define

```
public long scoreWorstCase(NGram guess)
```

as the maximum score of `guess` among all corpus' n-grams.

Similarly, she could take the view that the unknown key is one n-gram at random, in which case she would guess the n-gram that results in the smallest average corpus size. In `Corpus`, define

```
public long scoreAverageCase(NGram guess)
```

as the sum of scores of `guess` among all corpus' n-grams.

To guess an n-gram, the player has three public functions:

- `NGram bestWorstCaseGuess(NGram guess)` returns the best guess according to the worst-case (max) criterion.
- `NGram bestAverageCaseGuess(NGram guess)` returns the best guess according to the average-case (sum) criterion.
- `NGram bestGuess(ToLongFunction<NGram> criterion)` returns the best guess according to the custom `criterion`.

Example

Let's say a `Corpus` contains the n-grams "rebus", "redux", "route", and "hello" and the player guesses "route". For each n-gram in `Corpus`, `scoreWorstCase` calculates the score and returns the maximum.

1. The `scoreWorstCase` method first tries "rebus" against the "route" guess.
    a. Correct letter in the correct location: 'r'
    b. Correct but misplaced letters Misplaced: 'e', 'u'
    c. Absent: 'o', 't'

    The resulting filter leaves "rebus" and "redux", and the score is 2.

2. In the second step, "redux" is compared to "route".
    a. Correct letter in the correct location: 'r'
    b. Correct but misplaced letter: 'e', 'u'
    c. Absent: 'o', 't'

    The resulting filter leaves "redux" and "rebus", and the score is 2.

3. In the next comparison "route" is pitted against "route". All characters match, the resulting filter only leaves "route", and the score is 1.

4. Finally, "hello" is compared to "route".
    a. No letter is in the right location.
    b. Correct but misplaced letter: 'o' and 'e'
    c. Absent: 'r', 'u', 't'

    The resulting filter would only leave "hello" and the score is 1.

In the end, the maximum score is 2 when the key is "redux", so `scoreWorstCase` returns 2. Similarly, `scoreAverageCase` returns 6.

General Considerations

These classes may contain as many auxiliary private and package-private methods as you see fit, and additional package-private helper classes may be defined. However, any modification or addition to public classes and methods must be approved by the instructors at the discussion board.

You should write JUnit tests to make sure that your primary methods work as intended. However, we will revisit testing later on in the course, so extensive testing is not yet recommended. If you have never used JUnit, the JUnit module on canvas lists resources to get you started. Similarly, your code should have a reasonable number of comments, but documentation is going to be the topic of a future assignment. As a general guideline at this stage of the course, comments and tests should be like those accepted in CSDS 132.

Submission

Bring a copy to discussion to display on a projector. Additionally, submit an electronic copy of your program to Canvas. In addition to your code, include a README file explaining how to compile and run the code. The code should be handed in a zip, tar.bz2, or tar.gz archive. Archives in 7z cannot be accepted.