CSDS 293
Software Craftsmanship
2025 Spring Semester

Programming Assignment 2

Due at the beginning of your discussion session on

January 27-31, 2025

Reading

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

- Chapter 14 in Code Complete
- Items 28, 60, 62, and 64 in Effective Java
- Java's BigInteger and BigDecimal documentation (introduction only)
- Section 19.1 in Code Complete (excluding "Forming Boolean Expression Positively", "Guidelines for Comparison to 0", "In C-derived languages …", "In C++, consider ...")
- Section 15.1 ("Plain if-then Statements" only) in Code Complete
- Sections 17.1, 18.1 (excluding "Two Issues …"), and 19.4 in Code Complete ("Convert a nested if …", "Factor deeply nested code …", and "Use a more object-oriented approach" only)
- The object factory example on canvas

Programming

The objective of this and the next few programming assignments is to develop a system to solve a game of matchle. *Matchle* is similar but not identical to a well-known game published by a worldwide circulation newspaper.

Matchle

In a game of matchle, the player attempts to guess an n-letter word. After each attempt, the player receives information indicating when a letter occupies the correct position, is present in the answer but in a different position or is absent.

Suppose for example that the correct guess is "rebus" and the player guesses "route". The feedback would state that:

- The first character "r" is in the correct position
- The "u" and "e" are present in the correct word but in a different place, and
- The "o" and the "t" are absent.

At this point, the player can make another guess. For example, the player could guess "rules", which fixes the position of the first letter while attempting to find the position of the "u" and "e". However, he could also make a different reasonable guess. For example, the player could guess "yules", thereby forgoing the opportunity of making the correct guess (the first letter is an "r" and not a "y") to explore whether a "y" is present in any other position. At the end of programming assignment 5, your project will suggest the best guess.

The rest of the assignment will introduce the main classes and methods in the matchle solver.

Package

You should organize your project in a package. The package name is up to you: it could range from the simple ('matchle') to the detailed ('edu.cwru.<cwruid>.matchle').

NGrams

The `public final class NGram` formalizes the notion of a word containing *n* letters. The n-gram is at the core of the matchle game and of the guess system. It contains a `private final ArrayList<Character> ngram` that represents the word. For convenience, the word is also represented in a `private final Set<Character> charset`, which contains the same letters as in the `ngram`, is more easily searchable but omits the character order. It has a *private* constructor that initializes the private variables.

An `NGram` can be instantiated through two public static final methods:

- `NGram from(List<Character>)` returns a new `NGram` from a *copy* of the argument.
- `NGram from(String)` returns a new `NGram` from the characters in the argument.

The `NGram` has two public methods `get` and `size` that are delegated to the private `ngram`. The `get` method should return the `Character` at a given index. The `NGram` overrides `equals` and `hashCode` based on the contents of the private `ngram`.

Error Handling

Exceptions and Assertions

Error checking will be a topic of a future lecture. For the time being, the following guidelines should be implemented:

- If a method is *visible* outside the package (e.g., public methods in public classes), failed pre-conditions should cause the code to throw an appropriate exception. An exception can carry supplemental information that is passed as the constructor's argument(s).
- If a method is *invisible* from outside the package (e.g., private, or package-private methods), failed pre-conditions should be verified by an `assert`. An `assert` can specify supplemental information as the expression following the colon.

Supplemental Information

In both cases, the error handling should be supplemented by additional information. In Java, the supplemental information can take the form of one or more of the following:

- *Nothing*, if the source of the error is clear from other information already available (such as a stack trace).
- A `String` that outlines the reasons for the error. Strings should only be used if their usage conforms to Item 62 in the reading assignment.
- A `Throwable` that describes the cause of the error.

Null Arguments

A common occurrence of failed pre-conditions are arguments that are *null* even though null parameters are not expected or meaningful. Unless stated otherwise, you should assume through this series of assignments that null parameters constitute a failed pre-condition that necessitates error handling.

The Java Language Features module on canvas contains pointers on assertions, exceptions, and null parameter checking.

Exception

In the case of `NGram`, error handling involves a `public static final` nested class `NullCharacterException` that extends the standard Java language `Exception`. The `NullCharacterException` has a `private final integer index` that contains the location of the problematic character. The `index` has a public getter and the `NullCharacterException` has a public constructor that sets the `index` value. The `NullCharacterException`'s `serialVersionIUD` number can be set to any arbitrary value.

The `NullCharacterException` has a

```
public static final
List<Character> validate(List<Character> ngram)
```

that returns its argument, throws a `NullPointerException` if the argument is null, and throws an `IllegalArgumentException` whose cause is an `NullCharacterException` with the appropriate `index` if any character in the argument is null.

The two `NGram::from` methods should be modified to `validate` the argument before returning a new `NGram`.

Indexed Characters

In matchle, a simple but important concept is the pair of letters with the index where the letter appears. For example, in "hello", it can make a difference whether one refers to the first or the second instance of the letter "l". An `IndexedCharacter` is a public record that contains an `int index` and a `Character character`. For example, (2, "l") represents the first instance of "l" in "hello" and (3, "l") the second one. The `NGram` class implements three public methods:

- `boolean matches(IndexedCharacter c)` returns whether `c`'s character appears at the `c`'s index,
- `boolean contains(char c)` returns whether `c` appears anywhere in the n-gram, and
- `boolean containsElsewhere(IndexedCharacter c)` returns whether `c`'s character appears in the n-gram at an index different than `c`'s

NGram traversal

The `NGram` should be extended to support the following two methods through navigate through its indexed letter.

First, implement a `public Stream<IndexedCharacter> stream()` that gives the sequence of the n-gram's indexed characters.

Second, make `NGram` implement `Iterable<IndexedCharacter>`. To support the `Iterable`, create a `public final` nested class `Iterator` that implements the Java language `Iterator<IndexedCharacter>` and contains a `private int index`. The `Iterator` reads and updates the `index` values to implement the required `Iterator` methods.

Corpus

In matchle, both key and guesses must belong to a given dictionary. Furthermore, the dictionary words must have identical length.

Corpus

To represent a dictionary, create a `public final class Corpus` that implements `Iterable<NGram>` and contains an unmodifiable `private final Set<NGram> corpus` with a getter, and a private constructor. The `Corpus` delegates `size`, `contains`, `iterator`, and `stream` to the `corpus`. It implements the public methods:

- `Set<Ngram> corpus()` returns a *copy* of the private variable, and
- `int wordSize()` returns the common size of the n-grams in the corpus.

Builder

A `Corpus` is instantiated gradually by means of a builder class. Define a `public static final` nested class `Builder` that contains:

- `private Set<NGram> ngrams,`

- Private constructor that sets `ngrams`' value to the constructor's argument,
- `public static final Builder EMPTY`, which is a builder with no n-grams,
- `public static final Builder of(Corpus corpus)`, which returns a `Builder` whose n-grams are a copy of corpus'.

To incrementally add n-grams, the `Builder` implements two public methods. The first method is `Builder add(NGram)`, which adds a single n-gram or throws a `NullPointerException` if the n-gram is null. The second method is `Builder addAll(Collection<NGram>)`, which adds all the non-null n-grams in the collection or throws a `NullPointerException` if the collection is null. To validate the `ngrams`, the `Builder` implements a

```
public boolean isConsistent(Integer wordSize)
```

that returns whether all n-grams have the given `wordSize`. To build a `Corpus`, it implements `public Corpus build()`, which returns a new `Corpus` with the added n-grams (or `null` if the n-grams have inconsistent word size).

General Considerations

These classes may contain as many auxiliary methods and classes as you see fit. Helper methods should be private or package-private and helper classes should be package-private. Conversely, any modification or addition to public methods or classes 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 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. Additionally, comments should only be applied to the code sections that you feel are not self-documenting.

Canvas Resources

The module on Java Language Features contains a folder on useful Java features, such as enumerated types, regular expressions, and the optional class. A discussion board can be used to ask questions and post answers on this programming assignment.

Discussion Guidelines

The class discussion will focus on:

- High-level code organization
- The design and documentation of the implementation
- Straight-line code, conditional code

Your grade will be affected by the topics covered in this and in previous weeks. Your discussion leader may introduce material that has not been covered yet in the lecture. Although future topics will not contribute to your current grade, you are expected to follow your leader's advice in revising the assignment.

Submission

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. You can either bring a laptop to discussion to display your project on a projector or present your project from the canvas submission.

Note on Academic Integrity

It is a violation of academic integrity not only to copy another group's work but also to provide your code for other groups to copy including but not limited to posts on public github projects or social media.