Software Dev. & Problem Solving II GCIS-124

Iterators Assignment 5.2

Goals of the Assignment

The goal of this assignment is to practice creating ***iterable*** data structures so that they work with the `for each` loop. As always, you are expected to practice good software engineering, including unit testing and the Git workflow. Read this document ***<u>in its entirety</u>*** before asking for help from the course staff.

Activities

Part 1: Fibonacci

1. Create a folder named "iterators" under the unit05 directory for this assignment.

2. Create a class named `IterableFibonacci` that represents a Fibonacci sequence.

3. Add a constructor that declares two parameters of type `long` for the initial two numbers in the Fibonacci sequence.

4. Add the following methods:
   a. `add()` adds the next Fibonacci number to the sequence.
   b. `toString()` returns a nicely formatted string of its contents. For example:
   ```
   IterableFibonacci fib = new IterableFibonacci(2, 5);
   fib.add();
   fib.add();
   System.out.println(fib); // [2, 5, 7, 12]
   ```
   It's recommended to utilize the toString method of a data structure.
   c. `get(int index)` returns the Fibonacci number at the `index`. Using the above example:
   `get(0)` returns 2, `get(1)` returns 5, and so on.
   d. `length()` returns the length of the Fibonacci sequence.

5. Your Fibonacci class must be `iterable` to work with a for-each loop.
   a. `IterableFibonacci` should implement `java.util.Iterable<Long>`.

   b. Create a class named `FibonacciIterator` class that implements `java.util.Iterator<Long>` and use it in your Fibonacci class.

6. You are expected to declare all the required fields with their respective appropriate types.

7. Every method you write in your classes should run in *O(C)* time.

8. Write JUnit tests for each class and any non-trivial methods.

Part 2: File Reader

1. Create a class named `IterableReader`. Your reader should have the capability to read text files similar to the manner in which `java.io.BufferedReader` does, and should support the for-each loop, allowing it to iterate over each line in the file.

2. Add a constructor that declares a parameter for a file name. In the event that an IOException occurs, the constructor may throw it again.

3. The IterableReader class should not only be iterable but also an iterator itself. Additionally, the class should be AutoCloseable in order to work with the try-with-resource feature. To be clear, IterableReader implements the following interfaces:
   a. `Iterable<String>`
   b. `Iterator<String>`
   c. `AutoCloseable`

   Therefore, the following methods must be implemented.
   - `next()` reads a line of text and returns it. The method returns null if the end of the stream has been reached or an IOException occurred.
   - `hasNext()` returns true if the iterator has more elements, false otherwise.
   - `iterator()` returns the iterator for the current instance of IterableReader.
   - `close()` closes this stream.

4. Unit testing for Part 2 of this assignment is optional and can be omitted. Instead, create a main method where you manually test all the methods you wrote. For example,

```
  public static void main(String[] args) throws IOException {
        try (IterableReader reader = new
IterableReader("data/simple.txt");)
        {
            while (reader.hasNext()) {
                System.out.println(reader.next());
            }
        }
        try (IterableReader reader = new
IterableReader("data/simple.txt");)
        {
            for (String line : reader) {
                System.out.println(line);
            }
        }
    }
```

Submission Instructions

You must ensure that your solution to this assignment is pushed to GitHub *before* the start of the next lecture period.