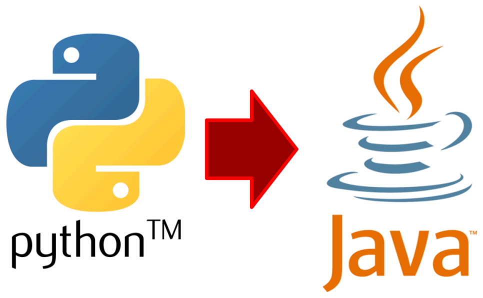
Software Dev. & Problem Solving II GCIS-124

File I/O and Exceptions Assignment 1.3



Goals of the Assignment

The goal of this assignment is to read from files and to practice exception handling in Java. Please read this document ***in its entirety*** before seeking help from the course staff.

- ***Read the assignment in its entirety***. Your questions about earlier parts of the assignment may be answered later in the description.
- The homework assignments build on the in-class activities; they are larger, more complex, and will take significantly more time to complete. Spend 20-30 minutes reviewing the lecture and in-class coding activities ***before*** you start the homework. If there are any activities that you cannot solve on your own, you will find the homework to be ***extremely challenging***. Seek help!
- While you should ***never*** share your assignment solutions with other students, It is OK to share code from in-class coding activities because every student already has access to the solutions. Feel free to share your code and ask questions in the `#lecture-questions` channel on Discord!
- Work in small increments; write only a few lines of code at a time and test it to make sure that it works before writing more.
- Start your assignment early, and plan to work during times that you know that ***help*** will be available. This includes:
    - Your instructor's office hours (see MyCourses)
    - Virtual mentoring (see the schedule pinned in `#waiting-room` on Discord)
    - The GCCIS Mentoring Center
    - SSE Mentoring hours (10 am-6 pm, M-F)
    - Women in Computing (WiC) Tutoring
    - The RIT Academic Success Center

Background

One way to determine whether or not an integer `n` is prime is through *brute force*; divide `n` by all values `k` where `1 < k < n`. You did something similar to this in a previous homework assignment. An alternative technique is to build a *Sieve of Eratosthenes*[1], an array of boolean or integer values where the value at each index `n` indicates whether or not `n` is prime. Consider the example table below where a value of `0` at an index `n` indicates that `n` *is* prime, and a value of `1` indicates that `n` is *not* prime.

n	prime	n	prime	n	prime	n	prime	n	prime
0	1	6	1	12	1	18	1	24	1
1	1	7	0	13	0	19	0	25	1
2	0	8	1	14	1	20	1	26	1
3	0	9	1	15	1	21	1	27	1
4	1	10	1	16	1	22	1	28	1
5	0	11	0	17	0	23	0	29	0

n	0	1	2	3	4	5	6	7	8
prime	1	1	0	0	1	0	1	0	1
n	9	10	11	12	13	14	15	16	17
prime	1	1	0	1	0	1	1	1	0
n	18	19	20	21	22	23	24	25	26
prime	1	0	1	1	1	0	1	1	1
n	27	28	29	30	31	32	33	34	35
prime	1	1	0	1	0	1	1	1	1

Given such a *sieve*, you can determine whether or not an integer `n` is prime in *constant time* by checking the value at its index in the array, for example:

```
    if(sieve[n] == 0) {
```

[1] Pronounced "Air-ah-toss-thuh-knees"

```
      System.out.println(n + " is prime!");
    } else {
      System.out.println("n + " is NOT prime.");
    }
```

The trick, of course, is making the sieve. Thankfully, that has been taken care of for you; sieves of several different sizes have been provided to you as text files in the data directory in your repository. The format of each file is fairly straightforward: the first line in the file is the *size* of the sieve, and the remaining lines contain 1s and 0s where 1 indicates a non-prime number, and 0 indicates a prime number. The number of columns in each file is variable. The following example is a sieve of size 100 where each line contains 10 digits.

```
100
1100101011
1010111010
1110111110
1011111011
1010111011
1110111110
1011111011
1010111110
1110111110
1111111011
```

Unfortunately, some of the files contain an error. It is up to you to find and fix those errors.

Activities

1. Take a few moments to examine the "`sieve_XXX.txt`" files in the data directory of your repository. Note that the first line of each file indicates the size of the sieve. The remaining lines in the file contain a series of 0s and 1s, each of which indicates whether or not the corresponding index in the array is prime (0) or not (1). The number of columns varies from file to file.

2. Create a new Java class in a file called "`SieveValidator.java`" and define a static method named "`readSieve`" that declares a parameter for a filename. The method should do the following:
    a. Open the file for reading.
    b. Use the first line of the file to create an integer array of the appropriate size. Hint: use the `Integer.parseInt(String)` method to convert the `String` into an `int`.
    c. Read the remaining lines in the file, and set the value at each index in the array to match each individual digit in the file. For example, the file `sieve_10.txt` contains the following text:

```
10
1100
1010
11
```

After reading it, your `int` array should contain the following values:

```
{1, 1, 0, 0, 1, 0, 1, 0, 1, 1}
```

d. Return the `int` array.
e. Your method should ***not*** throw any exceptions. If an exception does occur, return `null`.

3. Define a static method named "`repairSieve`" that declares a parameter for an `int` array. The method should do the following:
   a. Print a message indicating the size of the sieve being validated, e.g. "Validating a sieve of size 10…"
   b. Use the `Primes.isPrime()` method that you wrote in your previous assignment to validate that each index `n` correctly indicates whether `n` is prime (`0`) or not (`1`).
   c. If you find an index that is not correct, you should repair the sieve by changing the value. Print a message indicating the error, e.g. `"127 is incorrectly marked as NOT prime."`
   d. Return the number of errors that were found in the array.

   Note that, because arrays are reference types, changes that you make to the array inside the method will persist after the method returns. Therefore, there is no need for this method to return the array.

4. Create a JUnit unit test for your `SieveValidator` class. Write at least *two* tests for your `repairSieve` function to verify that it repairs errors in a Sieve of Eratosthenes. You do not need to write tests for the `readSieve` method.

5. Define a main method with the appropriate signature. Your method should do the following for each of the provided "`sieve_XXX.txt`" files in the data directory:
   a. Use the `readSieve` method to read the sieve into an `int` array.
   b. Use the `repairSieve` method to repair any errors.

   Here is some example output that does not necessarily match the files that have been provided to you. You should try to match the output format:

```
Validating sieve of size 10...
  Sieve contained 0 error(s).

Validating sieve of size 100...
  Sieve contained 0 error(s).
```

```
Validating sieve of size 5...
  2 is incorrectly marked as NOT prime.
  Sieve contained 1 error(s).

Validating sieve of size 55...
  15 is incorrectly marked as prime.
  Sieve contained 1 error(s).
```

Submission Instructions

You must ensure that your solution to this assignment is pushed to GitHub *before* the start of the next lecture period. See the course syllabus for the rubric that will be used to evaluate your submission.