Practice Coding Assignment 15 : Information Parser

This assignment is entirely optional and designed to give you practice writing code and applying lessons and topics for the current module.

This homework deals with the following topics:

- File I/O
- Basic text parsing
- Unit testing

The Assignment

In this assignment, you will build an information parser. The program will read and parse text from a file, process and extract the information needed, request new information from the user, and then write all of the information to a new file.

There are four main files in the program: *Main.java*, *InfoProcessor.java*, *MyFileReader.java*, *MyFileWriter.java*. The *MyFileReader* class will read the provided "info.txt" file and return the cleaned up contents of the file as an ArrayList<String>. The *InfoProcessor* class will process the array list provided by *MyFileReader* and extract the information needed. The *Main* class will request additional information from the user. And the *MyFileWriter* class will write all of the information to a new file "personal_info.txt".

The *Main* class will drive the entire program, and the "main" method calls have been provided for you. Nothing needs to be completed in the main method.

The program logic looks like this:

1. Using the *MyFileReader* class, read the "info.txt" file and get the cleaned up lines of text as an ArrayList<String>
2. Pass the ArrayList<String> from the *MyFileReader* class to the *InfoProcessor* class
    a. Extract the information needed, including: Course name, course id, and student id
    b. Store the information in a new ArrayList<String> called *linesToWrite*
3. Inside the "main" method in the *Main* class, request additional information from the user, including: Student name, favorite color, and favorite number
    a. Add this information to the ArrayList<String> *linesToWrite*
4. Write the ArrayList<String> *linesToWrite* to the "personal_info.txt" file using the *MyFileWriter* class

Implement all methods with "//TODO" in all classes.

MyFileReader Class

Instance Variables

The *MyFileReader* class has the following instance variable defined for you:

- String filename - Name of file being read

Constructor

The constructor in the *MyFileReader* class has been defined and **implemented for you**. You don't need to make any changes to the constructor in this class file.

The constructor has one parameter which represents the name of the file being loaded and read. The code in the constructor sets the value of the corresponding instance variable in the *MyFileReader* class.

Below is the code for the constructor in the *MyFileReader* class:

```
/**
 * Creates MyFileReader with given filename to read.
 * @param filename to read
 */
public MyFileReader(String filename) {
    this.filename = filename;
}
```

Methods

There is 1 method defined in the *MyFileReader* class that **needs to be implemented**:

- getCleanContent() - Opens the file specified by filename and reads the text line by line. Cleans the contents by trimming whitespace from the beginning and end of each line. Adds each line to an ArrayList<String> which is returned from the method.

Below is the code for the "getCleanContent" method in the *MyFileReader* class. For now, the method just returns null. Read the javadoc, which explains what the method is supposed to do. Then write your code where it says "// TODO". We have provided hints and example method calls to help you get started.

```
/**
 * Opens the file specified by filename and reads the text line by
 * line.
 * Cleans up each line by trimming whitespace from the beginning and
 * end of each line.
```

```
 * Adds each line to an ArrayList<String> which is returned from the
 * method.
 * If a line in the file is empty (does not contain any text), it's
 * skipped and is not added to the ArrayList<String>.
 *
 * @return list of lines with no empty spaces at the beginning or end
 * of each line
 */
public ArrayList<String> getCleanContent() {
    // TODO Implement method
    return null;
}
```

InfoProcesser Class

Instance Variable

The *InfoProcesser* class has the following instance variable defined for you:

- ArrayList<String> lines - List of lines to process

Constructor

The constructor in the *InfoProcesser* class has been defined and **implemented for you**. You don't need to make any changes to the constructor in this class file.

The constructor has one parameter which represents the list of lines to process. The code in the constructor sets the value of the corresponding instance variable in the *InfoProcessor* class.

Below is the code for the constructor in the *InfoProcesser* class:

```
/**
 * Creates InfoProcessor with given list of lines.
 * @param lines to process
 */
public InfoProcessor(ArrayList<String> lines) {
    this.lines = lines;
}
```

Methods

There are 4 methods defined in the *InfoProcessor* class that **need to be implemented**:

- getCourseName() - Gets the course name from the list of lines. Finds the line that starts with "Course:", then gets and returns the String on the very next line.
- getCourseId() - Gets the course ID from the list of lines. Finds the line that starts with "CourseID:", gets the String on the very next line, then casts the String to an int and returns it from the method.
- getStudentId() - Gets the student ID from the list of lines. Finds the line that starts with "StudentID:", gets the String on the very next line, then casts the String to an int and returns it from the method.
- getNextStringStartsWith(String str) - To be used by “getCourseName”, “getCourseId”, and “getStudentId” methods. Gets the String that follows the line that starts with the given String, then gets and returns the String on the very next line.

Again, each method has been defined for you in the *InfoProcessor* class, but without the code. See the javadoc for each method for instructions on what the method is supposed to do and how to write the code. It should be clear enough. In some cases, we have provided hints and example method calls to help you get started.

For example, below is the code for the “getCourseId” method in the *InfoProcessor* class. The method gets the course ID from the list of lines. For now, the method just returns 0. Read the javadoc, which explains what the method is supposed to do. Then write your code where it says “// TODO”. You’ll do this for the other methods in the *InfoProcessor* class as well.

```
/**
 * Gets the course ID from the list of lines.
 * First, finds the line that starts with "CourseID:".
 * Second, gets the String on the very next line, which should be the
 * course ID.
 * Third, casts the String to an int and returns it from the method.
 *
 * @return course ID
 */
public int getCourseId() {
    // TODO Implement method
    return 0;
}
```

MyFileWriter Class

Instance Variable

The *MyFileWriter* class has the following instance variable defined for you:

- String filename - Name of file being written to

Constructor

The constructor in the *MyFileWriter* class has been defined and **implemented for you**. You don't need to make any changes to the constructor in this class file.

The constructor has one parameter which represents the name of the file being written to. The code in the constructor sets the value of the corresponding instance variable in the *MyFileWriter* class.

Below is the code for the constructor in the *MyFileWriter* class:

```
/**
 * Creates MyFileWriter with given filename to write to.
 * @param filename to write to
 */
public MyFileWriter(String filename) {
    this.filename = filename;
}
```

Methods

There is 1 method defined in the *MyFileWriter* class that **needs to be implemented**:

- writeToFile(ArrayList<String> words) - Opens the file specified by filename and writes each word in the given list of words to the file. Each word is written to a new line.

Below is the code for the "writeToFile" method in the *MyFileWriter* class. Read the javadoc, which explains what the method is supposed to do. Then write your code where it says "// TODO". We have provided hints and example method calls to help you get started.

```
/**
 * Opens the file specified by filename and writes each String in the
 * given list of Strings to the file.
 * Each String is written to a new line.
 *
 * @param list of words to write to the file
 */
public void writeToFile(ArrayList<String> words) {
    // TODO Implement method
}
```

Unit Testing

In addition, you will write unit tests to test all method implementations in the program. Each unit test method has been defined for you, including some test cases. First make sure you pass all of the provided tests, then write **additional and distinct test cases** for each unit test method.

For example, in *MyFileReaderTest.java* we have defined 5 instances of *MyFileReader* for testing. We then initialize them inside of the "setUp" method, referencing the test text files in the constructors.

```
public class MyFileReaderTest {

    MyFileReader myFileReader1;
    MyFileReader myFileReader2;
    MyFileReader myFileReader3;
    MyFileReader myFileReader4;
    MyFileReader myFileReader5;

    @BeforeEach
    void setUp () {
        // original info.txt file
        this.myFileReader1 = new MyFileReader("info.txt");

        // file with multiple words per line
        this.myFileReader2 = new MyFileReader("test2.txt");

        // similar to info.txt file but with leading and trailing
        // whitespace
        this.myFileReader3 = new MyFileReader("test3.txt");

        // similar to info.txt file but with different info
        this.myFileReader4 = new MyFileReader("test4.txt");

        // similar to info.txt file but with blank lines in between
        //lines with information
        this.myFileReader5 = new MyFileReader("test5.txt");
    }
```

We have also defined a "testGetCleanContent" method for you (see below) which tests the "getCleanContent" method. It does this by using one of the test instances of *MyFileReader* and calling the "getCleanContent" method. Then it compares the result to a test ArrayList<String> of expected lines. Pass the tests provided then write additional tests where it says "// TODO".

```
@Test
public void testGetCleanContent() {
    ArrayList<String> actual = myFileReader1.getCleanContent();
    ArrayList<String> expected = new ArrayList<String>();

    // test original info.txt file
    expected.add("Course:");
    expected.add("MCIT_590");
    expected.add("Course_ID:");
    expected.add("590");
    expected.add("Student_ID:");
    expected.add("101");
    assertEquals(expected, actual);

    // TODO write at least 2 additional test cases using different
    // MyFileReaders
}
```

In *InfoProcessorTest.java* we have defined 4 instances of *InfoProcessor* for testing. We then initialize them, specifying unique ArrayLists in the constructors. In the "setUp" method, we add test data to the ArrayLists.

```
public class InfoProcessorTest {

    ArrayList<String> lines1 = new ArrayList<String>();
    ArrayList<String> lines2 = new ArrayList<String>();
    ArrayList<String> lines4 = new ArrayList<String>();
    ArrayList<String> lines5 = new ArrayList<String>();

    // sample file
    InfoProcessor infoProcessor1 = new InfoProcessor(lines1);

    // normal file
    InfoProcessor infoProcessor2 = new InfoProcessor(lines2);

    // file with different info
    InfoProcessor infoProcessor4 = new InfoProcessor(lines4);

    // another file with different info
    InfoProcessor infoProcessor5 = new InfoProcessor(lines5);
```

```
@BeforeEach
void setUp() {
    // sample file
    this.lines1.add("hello");
    this.lines1.add("world");
    this.lines1.add("MCIT");

    // normal file
    this.lines2.add("Course:");
    this.lines2.add("CIT590");
    this.lines2.add("CourseID:");
    this.lines2.add("590");
    this.lines2.add("StudentID:");
    this.lines2.add("101");

    // file with different info
    this.lines4.add("Course:");
    this.lines4.add("CIT 593");
    this.lines4.add("CourseID:");
    this.lines4.add("593");
    this.lines4.add("StudentID:");
    this.lines4.add("59876");

    // another file with different info
    this.lines5.add("Course:");
    this.lines5.add("OMCIT 596");
    this.lines5.add("CourseID:");
    this.lines5.add("596");
    this.lines5.add("StudentID:");
    this.lines5.add("01");
}
```

As an example of a test method in *InfoProcessorTest.java*, we have defined a "testGetCourseId" method for you (see below) which tests the "getCourseId" method. It does this by using one of the test instances of *InfoProcessor* to call the "getCourseId" method. Then it compares the return value with an expected int. Pass the tests provided then write additional tests where it says "// TODO". You'll do this for each unit test method noted in *InfoProcessorTest.java*

```
@Test
```

```
void testGetCourseID() {
    // test normal file
    int actual = infoProcessor2.getCourseId();
    assertEquals(590, actual);

    // TODO write at least 2 additional test cases using different
    // InfoProcessors

}
```

In *MyFileWriterTest.java* we have defined 4 instances of *MyFileWriter* for testing. We then initialize them inside of the "setUp" method, specifying unique output file names in the constructors.

```
public class MyFileWriterTest {

    MyFileWriter myFileWriter1;
    MyFileWriter myFileWriter2;
    MyFileWriter myFileWriter3;
    MyFileWriter myFileWriter4;

    @BeforeEach
    void setUp() {
        // file with multiple words per line
        this.myFileWriter1 = new MyFileWriter("test1_fw.txt");

        // similar to info.txt file
        this.myFileWriter2 = new MyFileWriter("test2_fw.txt");

        // similar to info.txt file but with personal info added
        this.myFileWriter3 = new MyFileWriter("test3_fw.txt");

        // similar to info.txt file but with different info and
        // personal info added
        this.myFileWriter4 = new MyFileWriter("test4_fw.txt");
    }
```

We have also defined a "testWriteToFile" method for you (see below) which tests the "writeToFile" method. It does this by creating a test ArrayList<String> of lines, and using one of the test instances of *MyFileWriter* to call the "writeToFile" method. Then it calls the "readWrittenFile" helper method to open, read, and test the outputted file and compares the

contents with the original test ArrayList<String> of lines. Pass the tests provided then write additional tests where it says "// TODO".

```
@Test
public void testWriteToFile() {

    ArrayList<String> actualLines = new ArrayList<String>();
    // Test file with multiple words per line
    actualLines.add("hello world");
    actualLines.add("Course Name and ID");
    actualLines.add("The quick brown fox jumps over the lazy dog.");

    // Write to file
    myFileWriter1.writeToFile(actualLines);

    // Read the written file to test its contents
    ArrayList<String> expectedLines =
readWrittenFile("test1_fw.txt");
    assertEquals(expectedLines, actualLines);


    // TODO write at least 2 additional test cases using different
    // MyFileWriters
    // Recommended: A test similar to info.txt file but with personal
    // info added
    // Recommended: A test similar to info.txt file but with
different
    // info and personal info added

}
```

Main Method

The "main" method in the *Main* class has also been defined and **implemented for you** (see example below). It takes care of calling all of the necessary methods, in order, in the program. Nothing needs to be completed in the main method.

```
/**
 * Main method to drive the program.
 * @param args
 */
```

```
public static void main(String[] args) {

    /*
     * Create new instance of file reader which reads the file
     * "info.txt"
     */
    MyFileReader fr = new MyFileReader("info.txt");

    /*
     * Create new instance of file writer which writes to the
     * file "personal_info.txt"
     */
    MyFileWriter fw = new MyFileWriter("personal_info.txt");

    /*
     * Clean lines of text passed from the file reader
     */
    ArrayList<String> lines = fr.getCleanContent();
```

Tips for this Assignment

In this assignment, some tips are given as follows:

- String Methods:
  - For removing leading and trailing spaces from Strings, consider the trim() method.
  - For checking if a String is empty (no content or having a length() == 0), consider the isEmpty() method.
  - See additional String methods here: https://docs.oracle.com/javase/7/docs/api/java/lang/String.html
- String to int conversion in Java:
  - Use "Integer.parseInt(str)"
    For example:

```
String str = "0";
int num = Integer.parseInt(str); //num is 0
```

- ArrayLists have many useful methods, such as get(), size(), etc. For a list of all methods available with ArrayLists, see: https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html

Submission

You have been provided with *Main.java*, *InfoProcessor.java*, *MyFileReader.java*, and *MyFileWriter.java*, which are the main program files. In addition, you have been provided with

info.txt, which is the file to load and read. You have also been provided with *InfoProcessorTest.java*, *MyFileReaderTest.java*, and *MyFileWriterTest.java* for testing. There are multiple test files to be used by *MyFileReaderTest.java* for loading and reading. These include: *test2.txt*, *test3.txt*, *test4.txt*, and *test5.txt*. Finally, there is a file "example_output.txt" which is an example of what the format of the outputted file from *MyFileWriter.java* should look like.

To complete the assignment, implement the methods in *InfoProcessor.java*, *MyFileReader.java*, and *MyFileWriter.java*, making sure you pass all the tests in *InfoProcessorTest.java*, *MyFileReaderTest.java*, and *MyFileWriterTest.java* respectively. (*MyFileReaderTest.java* will load and read *info.txt* and the additional test files noted above.) Then write **additional and distinct test cases** for each unit test method noted in *InfoProcessorTest.java*, *MyFileReaderTest.java*, and *MyFileWriterTest.java.* Do not modify the name of the methods in any of the files or the automated testing will not recognize it.

You will submit the following files for this assignment: *Main.java*, *InfoProcessor.java*, *MyFileReader.java*, *MyFileWriter.java*, *info.txt*, and *personal_info.txt*. You will also submit all of the testing files, including: *InfoProcessorTest.java*, *MyFileReaderTest.java*, *MyFileWriterTest.java*, *test2.txt*, *test3.txt*, *test4.txt*, *test5.txt*, and any other files you created for testing. Make sure your program and the unit testing files run without errors! Submit the completed program using the steps outlined in the assignment in Coursera.

Evaluation

This assignment is evaluated, so that you can assess how well you understand the material. However, the evaluation of the assignment will not affect your course grade.

Points

- *MyFileReader* class (5 pts)
  - getCleanContent() - 5 pts
    - Did you parse the file and store the clean content in the array list correctly?
- *MyFileWriter* class (5 pts)
  - writeToFile(ArrayList<String> words) - 5 pts
    - Did you write the file correctly?
- *InfoProcessor* class (9 pts)
  - getCourseName() - 2 pts
  - getCourseId() - 2 pts
  - getStudentId() - 2 pts
  - getNextStringStartsWith(String str) - 3 pts
- Did you include at least 2 additional distinct and valid test cases for each test method? Do all of your tests pass? (12 pts)
  - InfoProcessorTest => testGetCourseName() - 2 pts
  - InfoProcessorTest => testGetCourseID() - 2 pts
  - InfoProcessorTest => testGetStudentID() - 2 pts
  - InfoProcessorTest => testGetStudentID() - 2 pts
  - MyFileReaderTest => testGetCleanContent() - 2 pts
  - MyFileWriterTest => testWriteToFile() - 2 pts