Object-oriented programming and design

Lab #10

Using File I/O

Prerequisites, Goals, and Outcomes

Prerequisites: Before you begin this exercise, you need mastery of the following:

- *Java API*
  - Knowledge of the class `StringTokenizer`
- *File I/O*
  - Knowledge of file I/O
    - How to read data from a file
    - How to write data to a file

Goals: Reinforce your ability to use file I/O

Outcomes: You will master the following skills:

- Produce applications that read data from a file and parse it
- Produce applications that write data to a file

Background

This assignment asks you to implement two methods: one that reads employee data from a file and another that writes employee data to a file. The employee data contains basic information (ID, name, and salary) for a collection of employees.

Description

In this assignment, you will finish the implementation of `EmployeeFileIO`. iCarnegie will provide a test driver and the class `Employee`.

Class Employee

A complete implementation of this class is included in the student archive *`student-files.zip`*. Stop *now* and review its documentation:

- *`Employee.html`*. Documentation for class `Employee`

Class EmployeeFileIO

A partial implementation of this class is included in the student archive *`student-files.zip`*. You should complete the implementation of the following methods:

- *`public static ArrayList<Employee> read(String fileName)`*
- *`    throws FileNotFoundException,`*
- *`           IOException,`*
- *`           NoSuchElementException,`*
- *`           NumberFormatException`*

  This method creates an `ArrayList` of `Employee` objects from a file that contains employee data.

- *`public static void write(String fileName, ArrayList<Employee> arrayList)`*

- *throws IOException*

  This method creates a file of employee data from an `ArrayList` of `Employee` objects.

Class `TestEmployeeFileIO`

This class is a test driver for `EmployeeFileIO`. It contains test cases for each method in `EmployeeFileIO`. A complete implementation is included in the student archive *student-files.zip*. You should use this class to test your implementation of `EmployeeFileIO`.

Your implementation of method `read` is tested by comparing the `ArrayList` returned by your implementation against an `ArrayList` returned by the iCarnegie implementation. In the same way, your implementation of method `write` is tested comparing the file produced by your implementation against a file produced by the iCarnegie implementation.

Class `TestHelper`

This class contains auxiliary methods used by the test driver: a method for comparing two `ArrayList` objects and a method for comparing two files. A complete and compiled implementation is included in the student archive *student-files.zip*.

Review its documentation and become familiar with it:

- *TestHelper.html*. Documentation for class `TestHelper`

Files

The following files are needed to complete this assignment:

- *student-files.zip* — Download this file. This archive contains the following:
  - Class files
    - *TestHelper.class*
  - Documentation
    - *Employee.html*
    - *TestHelper.html*
  - Java files
    - *Employee.java*. A complete implementation
    - *TestEmployeeFileIO.java*. A complete implementation
    - *EmployeeFileIO.java*. Use this template to complete your implementation.
  - Data files used by the test driver
    - *empty.txt*. An empty file
    - *employees.txt*. A file with employee data

Tasks

Implement the methods `read` and `write` in class `EmployeeFileIO`. Document your code using Javadoc and follow Sun's code conventions. The following steps will guide you through this assignment. Work incrementally and test each increment. Save often.

1. **Extract** the files by issuing the following command at the command prompt:

   ```
   C:\>unzip student-files.zip
   ```

2. **Test** each method as soon as you finish writing it by issuing the following command at the command prompt:

   ```
   C:\>java TestEmployeeFileIO
   ```

3. **Implement** the method `read`: It begins by creating an empty `ArrayList` and a BufferedReader object to read data from the specified file. It then proceeds to read each line in the file. After it reads a line, it extracts the ID, name, and salary of an employee, creates an `Employee` object, and adds the new object to the end of the `ArrayList`. When all data has been read, it returns the `ArrayList`.

Use `BufferedReader.readLine` to read the data in the file. Use `java.util.StringTokenizer` to parse the data.

You can assume that every line in the file contains the data for exactly one employee in the following format:

`ID_name_salary`

where:

- *ID* is an integer that represents the ID of the employee.
- *name* is a String that represents the name of the employee.
- *salary* is a double that represents the salary of the employee.

The fields are delimited by an underscore (_). You can assume that the fields themselves do not contain any underscores.

The method `read` should *not* contain try-catch blocks for the following exceptions. This requirement will simplify the code.

- *`FileNotFoundException`*. Thrown by the `BufferedReader` constructor if the specified file does not exist.
- *`IOException`*. Thrown by `BufferedReader.readLine` if an I/O error occurs.
- *`NoSuchElementException`*. Thrown by `StringTokenizer.nextToken` if the specified file contains incomplete data.
- *`NumberFormatException`*. Thrown by `Integer.parseInt` and `Double.parseDouble` if the specified file contains invalid data.

4. **Implement** the method `write`: It first creates a PrintWriter object for writing data to the specified file (if the file does not exist, one will be created). It then writes the ID, name, and salary of each employee in the specified `ArrayList` to the specified file. Every line in the file should contain the data for exactly one employee in the following format:

   `ID_name_salary`

   where:

   - *ID* is an integer that represents the ID of the employee.
   - *name* is a String that represents the name of the employee.
   - *salary* is a double that represents the salary of the employee.

   The fields are delimited by an underscore (_).

   The order of the employees in the file should match the order of the employees in the `ArrayList`. If the specified file exists, its contents should be erased when it is opened for writing.

The method `write` should *not* contain a try-catch block for the following exception. This requirement will simplify the code.

- *`IOException`*. Thrown by the `FileWriter` constructor if the specified file could not be found or created.

Submission

Upon completion, submit **only** the following.

1. `EmployeeFileIO.java`