Java Final Exam: Expense Management System

This exam is due Monday, December 19th @ 11:59pm. You may take as long as you need on the exam up until the deadline.

YOU CANNOT:

- Work with someone else on this exam.
- Copy from someone else's exam.
- Use StackOverflow or any search engine, such as Google.com, for any reason (other than to go to the course exam or approved resources).
- Use an internet search for keywords in the exam. For example, do not Google "loading a .csv file in Java".
- Discuss the exam with other students until cleared by the instructor to do so. Other students have not yet taken the exam, and you CANNOT provide academically dishonest assistance to them.

YOU CAN:

- Reference any material from the course or recitation that can be accessed in Canvas. This includes videos, readings, slides, code samples, homework assignments, quizzes, and practice coding exercises.
- Reference approved online Java documentation such as:
    - https://docs.oracle.com/en/java/
    - https://www.w3resource.com/java-tutorial/
    - https://www.baeldung.com/

Use of Ed Discussion/Office Hours as it Relates to the Exam

If you have any clarification questions regarding the exam, please make a **private post** on Ed Discussion. We will keep a pinned thread going of commonly asked questions, so please be sure to check that before posting your question

The Assignment

This Java exam will involve implementing an *expense management system*. You will be provided with the skeleton of the program and you will have to implement the methods and write the unit tests. The design of the program has been set up for you.

In this system, expense files can be loaded, parsed, and managed in the context of the system. Expenses (represented by a single Expense class and are combined into the plural Expenses class) are loaded and stored in a management system (represented by an *ExpenseManagement* class). An *ExpenseFileReader* class will take care of initially loading the expense files and do some basic clean-up with the assistance of an *ExpenseFileParser* class.

Overall, the program will load an expense file (expenses.txt) with *ExpenseFileReader* and *ExpenseFileParser*, create *Expense* objects stored in an *Expenses* collection, then store them in the *ExpenseManagement* system. The required methods are already called in the *main* method in each section of the exam.

Exam Instructions

To help guide you through implementing the program, **the exam has been split into 4 sections**. For each section, there will be 2 tasks. The first task will be to complete the required methods in one java file and the second task will be to complete the unit testing for those methods in another java file. You will be provided with all of the necessary java files to complete the tasks in each section.

You are not required to complete the sections in a specific order. This is merely set up as a guide.

Exam Overview

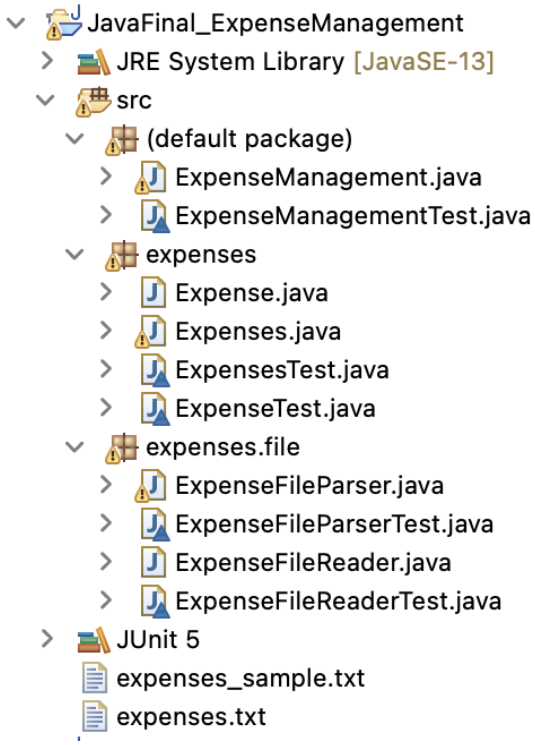
For each section on the exam, the first task is to complete the required methods in the given class file. Then compile and run your code.

- Javadocs have already been provided
- You can create any number of helper methods, with appropriate names and javadocs
- Name your variables appropriately
- Add brief comments to all non-trivial lines of code
- For convenience, all of the required methods in the given class file are already called in its respective *main* method (follow along to learn the sequence of the methods). (Note, you are not required to run the *main* method in the given class file.)

The second task is to test your code by running (and passing) the provided test cases in the given test class file. (**Do not modify any of the provided test cases!)** Then **write additional test cases for each test method as noted** and make sure they pass as expected. (Note, you do not have to write unit tests for any helper methods you write.)

- **The provided test cases are just SOME of the unit tests used for the autograded portion of this assignment. There will be more tests for the full autograded portion.**
- You do not have to write unit tests for any helper methods you write.

The final structure of the project in Eclipse should look like this:



Exam Outline

- **Section 1**
  - Task 1: ExpenseFileReader.java
    - Complete the loadExpenses method in the ExpenseFileReader class. Compile and run your code.
      - *Note: You can run the main method in this class.*
  - Task 2: ExpenseFileReaderTest.java

        - Test your code by running (and passing) the provided test cases in the ExpenseFileReaderTest class. Write additional test cases as noted in the testLoadExpenses method and make sure they pass as expected. At the end, the testLoadExpenses method should **have a total of 1 distinct new test case.**

- **Section 2**
    - Task 1: ExpenseFileParser.java
        - Complete the parseExpenses method in the ExpenseFileParser class. Compile and run your code.
            - *Note: You can run the main method in this class.*
    - Task 2: ExpenseFileParserTest.java
        - Test your code by running (and passing) the provided test cases in the ExpenseFileParserTest class. Write additional test cases as noted in the testParseExpenses method and make sure they pass as expected. At the end, the testParseExpenses method should **have a total of 1 distinct new test case.**

- **Section 3**
    - Task 1: Expense.java and Expenses.java
        - Complete the equals method in the Expense class, then the getMonthlyExpenses, addExpenses, and addExpense methods in the Expenses class. Compile and run your code.
            - *Note: You can run the main method in the Expenses class.*
    - Task 2: ExpenseTest.java and ExpensesTest.java
        - Test your code by running (and passing) the provided test cases in the ExpenseTest class. Then write additional test cases as noted in the testEquals method and make sure they pass as expected. At the end, the testEquals test method should **have a total of 1 distinct new test case.**
        - Test your code by running (and passing) the provided test cases in the ExpensesTest class. Then write additional test cases as noted in the testAddExpense and testAddExpenses methods and make sure they pass as expected. At the end, each test method should **have a total of 2 distinct new test cases.**

- **Section 4**
    - Task 1: ExpenseManagement.java

        - Complete the getMonthlyExpenses, getMonthlyExpenses, getTotalMonthlyExpenses, and getMostExpensiveMonth methods in the ExpenseManagement class. Compile and run your code.
            - *Note: You can run the main method in this class.*
    - Task 2: ExpenseManagementTest.java
        - Test your code by running (and passing) the provided test cases in the ExpenseManagementTest class. Then write additional test cases as noted in the testGetMonthlyExpensesIntListOfExpense, testGetMonthlyExpensesStringListOfExpense, testGetTotalMonthlyExpenses, and testGetMostExpensiveMonth methods and make sure they pass as expected. At the end, each test method should **have a total of 2 distinct new test cases.**

Submission

You will submit your entire project in a .zip file. Include your "src" folder with all of your code , as well as any data files outside of the "src" folder. Do not remove the package declarations in each file and keep the files in their respective package subdirectory as specified above.

Evaluation

1. Did you implement the individual methods correctly? *– 30 points (autograded via a combination of provided and hidden unit tests)*
    a. Can you get the monthly expense data successfully?
    b. Does your program successfully load and parse the expenses.txt files and store it properly?
    c. Does it correctly extract the expense information
2. Did you write good unit tests? *– 15 points*
    a. Did you make sure each test method has the required number of test cases?
    b. Does your program pass all of your own tests?
3. Coding Style (this includes but is not limited to the following) – 5 points
    a. All non-trivial lines of code should have clear and descriptive comments
    b. Variables should be named based on the information they store
    c. If applicable, helper methods should be named based on what they do (and include javadocs)
    d. Remove unused or commented out variables/lines of code
    e. Variables and methods adhere to camelcase conventions