CS 520
Homework 1
Design & Manual Reviewing

Due: **Thursday October 2, 11:59 PM (a little before midnight)** via Gradescope.

You can choose to do the homework individually or in a pair. If working in a pair, this homework assignment is a group submission, meaning each pair submits one solution, and both members receive the same grade. You may collaborate with your partner but not with other pairs. Submissions from different pairs must be independent. Late assignments may be accepted under extenuating circumstances.

Overview and Goal

The goal of this assignment is to analyze, document, and redesign an Expense Tracker App following the Model-View-Controller (MVC) architecture pattern, emphasizing software engineering principles.

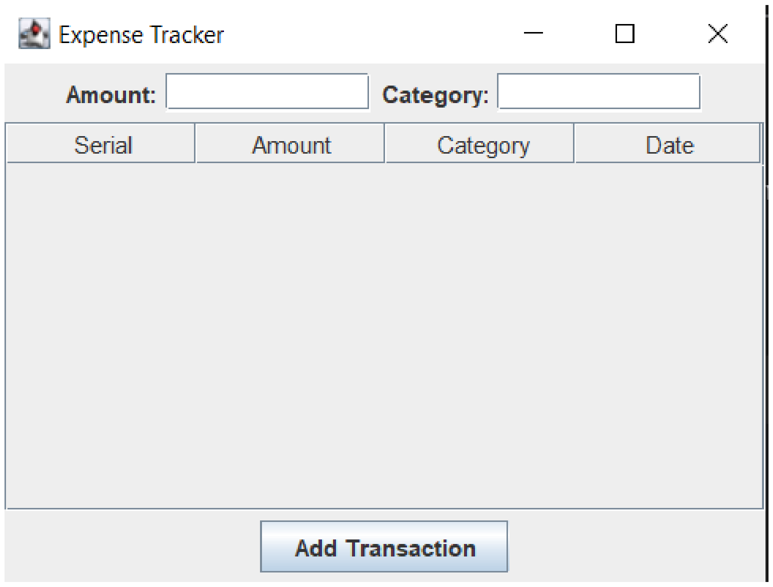

Figure 1: Screenshot of the ‘Expense Tracker‘ UI

This quick-and-dirty implementation satisfies some key non-functional requirements but violates others. It needs a major design overhaul. In contrast to the current version, your implementation should try to improve the following non-functional requirements: understandability, modularity, extensibility, testability, and debuggability.

How to get started

1. Clone the repository with the following command:

```
git clone https://github.com/CS520-Spring2025/hw1_baseline
```

2. Read the provided *README* in the folder and use the commands to document, compile, test, and run the application.
3. Familiarize yourself with the original application source code contained in the *src* and *test* folders: src/ExpenseTrackerApp.java, src/ExpenseTrackerView.java, src/Transaction.java, test/TestExample.java.

By the end of the semester, your version of the application must adhere to:

- the MVC architecture pattern
- OO design principles and patterns
- Best programming practices

Part 1: Manual Review (20 Points)

You are expected to manually review the original version of the application focusing on the expected behavior (e.g., extensibility, testability) instead of coding style (e.g., amount of whitespace, if vs switch statement). Please refer to the `Readme` file for an example pattern on identified satisfaction and violation of the non-functional requirements. Analyze the provided application and identify **non-functional requirements**:

1. **Two satisfied non-functional requirements:**
    - Briefly describe the requirement (e.g., modularity, testability).
    - Provide an example from the code (class, method, field, or screenshot).
2. **Two violated non-functional requirements:**
    - Briefly describe the requirement.
    - Provide an example from the code (class, method, field, or screenshot).
    - Explain why it is a violation.
    - Suggest a possible fix.

Part 2: Modularity (24 Points)

2.1 Model View Controller (MVC) Architecture Pattern (6 points)

Identify the components in Figure 2 (Expense Tracker UI Components):

- Component A: View, Controller, or both? Briefly explain what is being visualized and/or what user interaction is being provided.
- Component B: View, Controller, or both? Briefly explain what is being visualized and/or what user interaction is being provided.

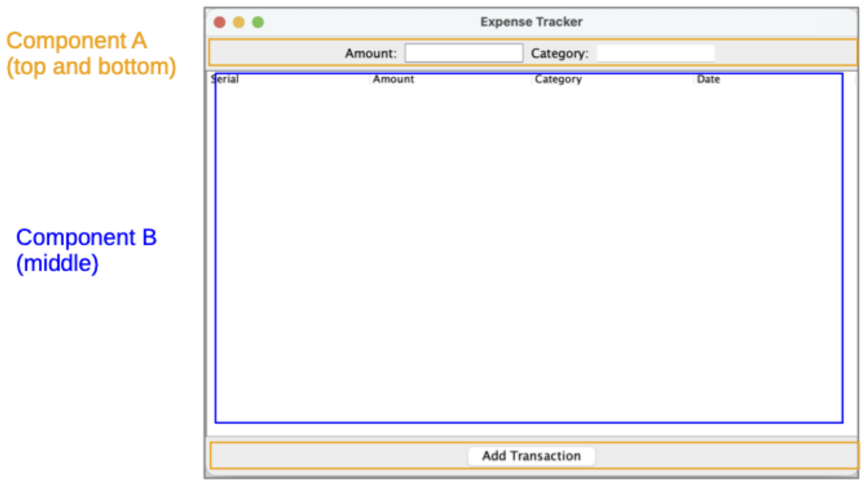

Figure 2: Main components of the ‘Expense Tracker‘ UI

2.2 Open-Closed Design Principle (15 points)]

The “‘ExpenseTrackerView.java“‘ currently manages the data related to transactions. The “‘ExpenseTrackerView“‘ class uses the “‘Transaction“‘ class to store and manage individual transactions in the transactions list. Your implementation should establish data encapsulation and immutability of the “‘Transactions“‘ data.

- Apply encapsulation for the list of transactions.
- Apply immutability on the list of transactions when the getter method is invoked.
- Apply changes to the “‘Transaction“‘ class to prevent external data modification. Ensure information hiding for the declared fields. Some methods may be necessary to remove to make the class immutable.

The above should be **committed** to your git repository.

Part 3: Understandability - Documentation (12 Points)

- Update the `README.md` file to briefly describe the supported features.
- Show the API for a given class, meaning its set of methods, or a specific method, meaning its API documentation comment, in an Integrated Development Environment (IDE). For example, you could take a screenshot of a class outline view or a method’s Javadoc view.
- Generate the API documentation (with Javadoc) and commit it to the `jdoc` folder.
- Maintain incremental commit messages documenting progress.

Part 4: Code Modification - Input Validation (12 Points)

The current code doesn’t provide validation for the input data types. You should create a file named `InputValidation.java` to validate the input fields and **commit it**: `amount` and `category`. Modify the application to include input validation:

- Create `InputValidation.java` to validate:
  - `amount`: Must be a valid number greater than 0 and less than 1000.
  - `category`: Must be one of: `"food", "travel", "bills", "entertainment", "other"`.
- Update `ExpenseTrackerApp.java` to integrate validation before processing transactions.

  We will check that the application successfully compiles and runs.

Part 5: Extensibility - Adding a Filtering Feature (15 Points)

Describe how to implement a **filtering feature** in the Expense Tracker, allowing transactions to be filtered by:

- Category
- Amount range
- Date

For each, specify:

- Which fields and methods need modification/addition.
- Briefly explain the necessary changes.

You should provide a description of how to add *filtering* on the list of added transactions. You can think of filtering based on *category* types, *amount* or by *date*. For simplicity, think about applying one filter at a time. Your description should identify the set of fields and methods that need to be modified or introduced to support the extension. For each identified field or method, briefly describe the necessary changes to support the extension.

Deliverables (16 points)

Submit a single **ZIP archive** named `hw1_expensetracker.zip` containing:

1. Updated Expense Tracker source code (`src` & `test` folders).
2. Generated Javadoc documentation (`jdoc` folder).
3. `ExpenseTrackerView.java` (modified).
4. `Transaction.java` (modified)
5. `InputValidation.java` (newly created).
6. `HW1_answers.pdf` (contains manual review, UI analysis, and proposed extension).
7. Git commit log showing incremental progress.

The *expense_tracker* folder with all the updated source files and test cases of your application. The git log should have a set of coherent commits showing your work, not a single version of the code. Additionally, the hw1 file (including the manual review, proposed extension, and UI analysis) needs to be submitted as a PDF file.

Upload to Gradescope.