CS 520
Homework 1
Program understanding

Due: **September 27, Wednesday, 11:59 PM (a little before midnight)** via Gradescope.

Each programming pair may work with others on this assignment but each pair must make their own submission. Any software development artifacts (e.g, natural language documents, source code) should be individual to that pair, not created jointly with others, and written in the pair's own words and style.

Late assignments will be accepted for extenuating circumstances.Please mention the late days used in this homework in the `hw1.pdf` file. There will be 100 points in total.

Overview and goal

The goal of this assignment is to manually review and redesign an Expense Tracker App, according to the model-view-controller (MVC) architecture pattern.

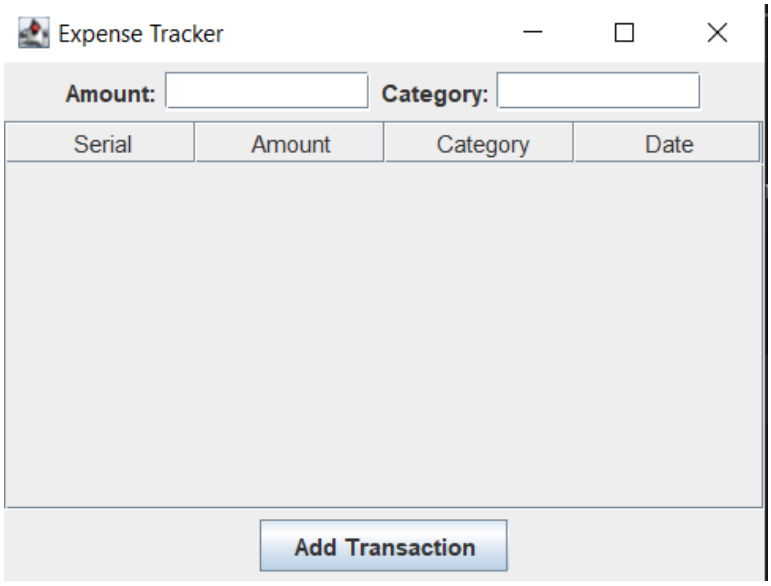

Figure 1: Screenshot of the 'Expense Tracker' UI

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-Fall2023/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

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. **Identify the following:**

- two examples of non-functional requirements that are satisfied
- two examples of non-functional requirements that are violated

For the satisfied non-functional requirements, please follow the pattern:

- Brief summary of the non-functional requirement (a few keywords)
- An illustrative example from the code. The example should either specify a code element name (e.g., class, method, field) or be a screenshot. Line numbers should not be used.

For the violated non-functional requirements, please follow the pattern:

- Brief summary of the non-functional requirement (a few keywords)
- An illustrative example from the code. The example should either specify a code element name (e.g., class, method, field) or be a screenshot. Line numbers should not be used.
- Explanation of this issue (could refer to general principles or poor design choices with respect to the desired non-functional requirement)
- How to fix the issue (a few sentences)

Modularity: MVC architecture pattern [25 points]

In Figure 2, identify the following components from the Application's perspective in the UI (not Java.swing perspective):

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

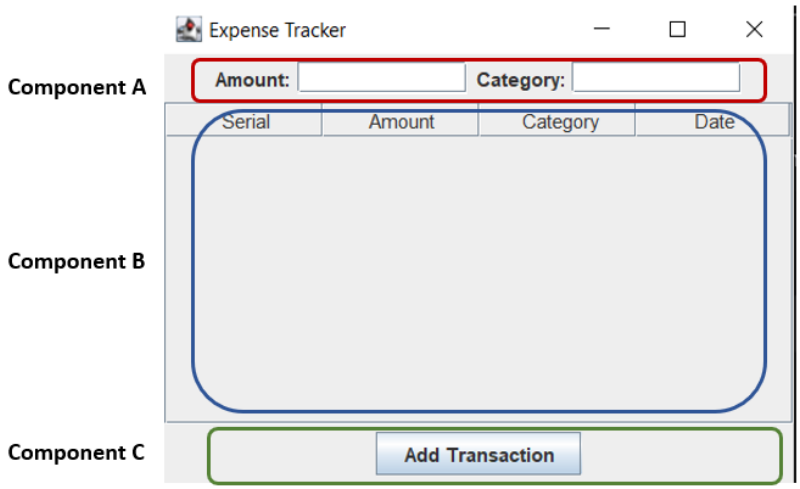

Figure 2: Main components of the ‘Expense Tracker‘ UI

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

Identify the original application source code (including all classes, fields, and methods) corresponding to the:

- 2.2.1) Model
- 2.2.2) One view (from the 3 components listed above)
- 2.2.3) One controller (from the 3 components listed above)

Code Modification [20 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`. Some hints are as follows:

- The amount should be greater than 0 and less than 1000.
- It should be a valid number, check for invalid inputs of other data types.
- The category should be a valid string input from the following list: ”food”, ”travel”, ”bills”, ”entertainment”, and ”other”.
- Update the `ExpenseTrackerApp.java` with the input validation steps for adding transaction.

Understandability: Documentation [Approximately 10 points]

You should update the documentation including:

1. You should generate the javadoc (contained in the jdoc folder) and **commit it**.
2. You should have incremental commit messages.

Extensibility: Proposed extension [15 points]

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 [Approximately 10 points]

For your submission, we will be checking the following:

1. The **HW1_answers** file including the manual review and the proposed extension (either text or PDF file) is included.
2. The jdoc folder and its contents are included
3. The app compiles and runs
4. The test suite compiles, runs, and all test cases pass
5. `InputValidation.java` file.

The submission, via Gradescope, must be a single archive (.zip) file named `hw1_expensetracker`, containing:

1. 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 and proposed extension) needs to be submitted as either a text or PDF file.