# CS 520
# Homework 3
# Implementation & Debugging

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

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

Late assignments will be accepted for extenuating circumstances.

There will be 100 points in total.

## Overview and goal

The goal of this assignment is to redesign and implement features for the `Expense Tracker App`, to improve adherence to non-functional requirements (e.g., understandability, modularity, extensibility, testability, debuggability), design principles/patterns (e.g., open-closed principle, Strategy design pattern), and best practices.

This implementation applies the MVC architecture pattern. In contrast to the current version, your implementation should support possible extensions to satisfy the Extensibility. Additionally, your implementation should enable individual components to be tested in isolation. The implementation satisfies some best practices but violates others. You are expected to clone the existing repository and keep your implementation under version control, using the cloned repository. You will submit your repository to us, so you should make coherent and atomic commits, and use descriptive log messages.

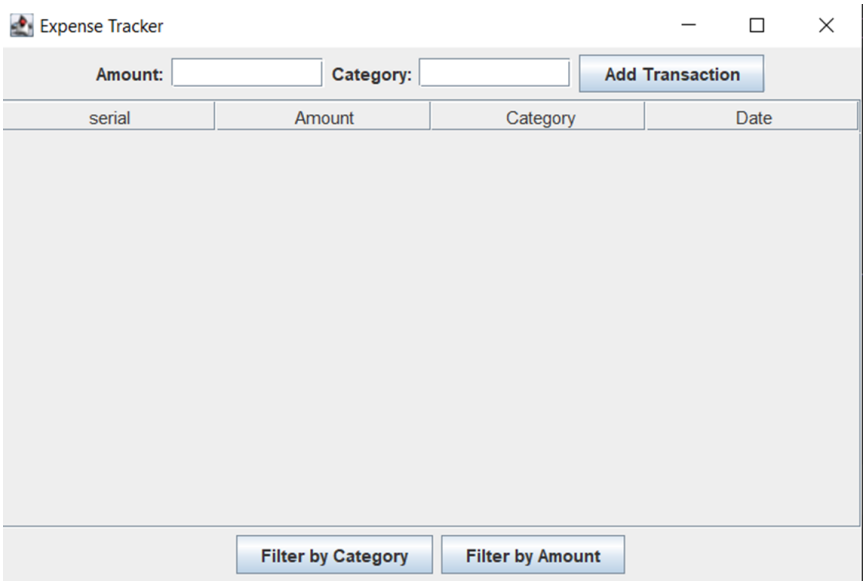


Figure 1: Screenshot of the ‘Expense Tracker‘ UI

### How to get started

1. Clone the repository with the following command:

```
git clone https://github.com/CS520-Fall2025/hw2-solution
```

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

## Understandability: Documentation [15 points]

You should update the documentation including:

1. You should update the README file to document any new functionality and **commit it**.
2. Add API comments to any new classes/interfaces and **commit them**.
3. You should generate the javadoc (contained in the jdoc folder) and **commit it**.
4. You need to include the git log. Each git commit must show the author, timestamp, and message. There should be incremental commits.

### Usability: Export to CSV file [25 points]

Several of your users submitted a new feature request to export their expense tracker history to a CSV (Comma Separated Value) file. Your design and implementation need to apply:

- the MVC architecture pattern
- UI design laws (e.g., provides help)
- OO design principles (e.g., open-closed principle)
- best practices (e.g., no magic strings).

You are responsible for designing and implementing this new feature where:

- The user should be able to specify the output file name. Your implementation must validate that the file name is non-empty and ends with ".csv", and must notify the user (e.g., via a dialog box) when the input is invalid.
- For a valid output file, the first line should contain the column headers. Each remaining line should contain a transaction.

Your user acceptance test must be written as a numbered list of steps a user could follow:

1. Initial condition: a screenshot of the UI with at least one transaction added.
2. Steps to be performed: a numbered list describing how a user exports the CSV (e.g., enter a file name, click Export).
3. Success end condition: a screenshot of the generated CSV file showing the exported rows.

### Debuggability: Debugger [20 points]

You should select a Java Integrated Development Environment (e.g., Eclipse, VSC). Your goal is to demonstrate the use of breakpoints and program-state inspection to show the correctness of part of the existing filtering logic.

To complete this section, do the following:

1. Add a breakpoint to your `applyFilter` method.
2. Run the Expense Tracker App in debugging mode to hit that breakpoint.
3. Perform the setup of the `testFilterByCategory` test case and inspect the program state, which corresponds to the preconditions of that test case.
4. Apply the category filter that includes (shows) the food related transactions you just added and inspect the program state, which now corresponds to the postconditions of the test case.

You must include the following three (3) debugger screenshots:

1. IDE showing the breakpoint for the `applyFilter` method.
2. Before applying the `applyFilter` method, the debugger's variables pane must show the preconditions program state (e.g., the entire `Transactions List` and `Total Cost`).
3. After applying the `applyFilter` method, the debugger's variables pane must show the postconditions program state (the displayed/filtered view and/or UI widgets should now only show the food related transactions).

We should be able to clearly see the difference in program state between screenshots (2) and (3).

## UI Design Principle: Undo Functionality [20 points]

One good UI design principle is to provide `undo` functionality. Here is the proposal for a simple `undo` in this app:

- A user selects a row to delete its corresponding Transaction.
- The deletion must update the `Transaction List` appropriately.
- The deletion also must update the `Total Cost` appropriately.

You are responsible for providing a written design that describes the above simple undo. You must describe the necessary changes to the `Model, View, and Controller`. Your design should also briefly explain how it satisfies relevant non-functional requirements such as understandability, extensibility, or testability. You do not need to implement this section, submit it as a written design plan as a plain-text document `undo.txt` or PDF document `undo.pdf`.

## Deliverables [Approximately 20 points]

1. Programming Part:
    - Generated javadoc in a folder `jdoc/`
    - Git log (`gitlog.txt` or `gitlog.pdf`) with meaningful commits
    - Application compiles and runs successfully
    - Test suite compiles, runs, and all test cases pass
2. Written Part:
    - User acceptance test case from the Usability section
    - Three screenshots from the Debuggability section
    - UI design principle plan

### Submission Format

For the programming submission, submit a ZIP file named `homework3_expensetracker.zip`.

Inside the ZIP, there must be a top-level folder named `expense_tracker/`, since the autograder depends on this name. The required internal project structure is:

```
expense_tracker/
    src/                # All source files
    test/               # JUnit test files
    lib/                # The junit library
    jdoc/               # Generated Javadoc
    build.xml
    README.md           # Updated README
    gitlog.txt or .pdf
```

For the written submission, submit a separate PDF named `answers.pdf` containing:

- the export user acceptance test case
- the filter debugger screenshots
- the undo design plan

Submit via Gradescope-programming and Gradescope-Written