

CS 520

Homework 2

Implementation & Testing

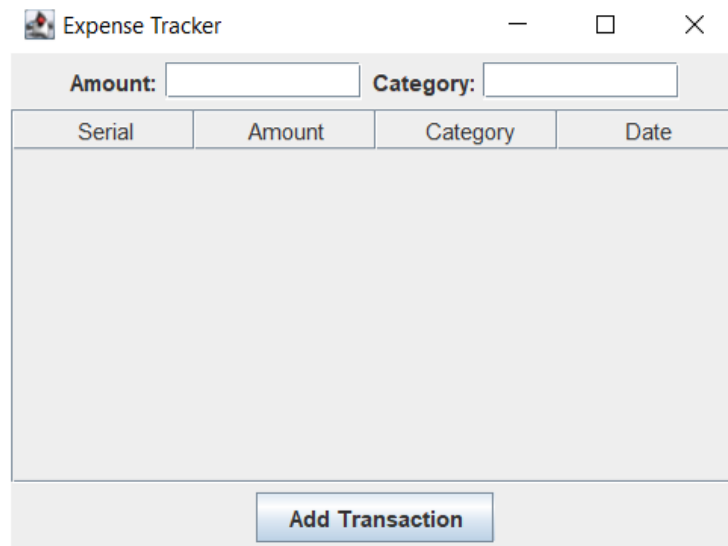
Due: **Thursday, April 10, 11:59 PM (a little before midnight)** via [Gradescope](#).

Each homework group (either an individual student or pair of students) may work with others on this assignment but each group must make their own submission. Any software development artifacts (e.g, natural language documents, source code) should be unique to that group, not created jointly with others, and written in the group's 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), 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 (at least for the first 3 sections below), and use descriptive log messages.



Serial	Amount	Category	Date
--------	--------	----------	------

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-Spring2025/hw2.git
```

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

Understandability: Documentation [12 points]

1. Update the README file to document any new functionality and **commit it**.
2. Generate the javadoc (contained in the *jdoc* folder) and **commit it**.
3. You need to include the git log. Each git commit must show the author, timestamp, and message. There should be incremental commits.

Modularity: Open-Closed Principle [12 points]

This version of the application has started to apply the MVC architecture pattern. The `ExpenseTrackerModel.java` manages the data related to transactions. The `ExpenseTrackerModel` 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. Remove setters and hide fields as necessary.

The necessary code modifications should be **committed** to your git repository.

Extensibility: Strategy Design Pattern [20 points]

Implement the `filter` feature for attributes *amount* and *category* by following a strategy design pattern. This allows the encapsulation of each filter algorithm into reusable classes. Users can filter by either *amount* or *category* at a time. The input validation that was developed in homework 1 for *amount* and *category* should be applied on the filters as well. Display only the transactions that match the filter.

- Create a `TransactionFilter` interface: Defines `List<Transaction> filter(List<Transaction>)` method
- Implement `CategoryFilter` and `AmountFilter` strategies.
- Update the `ExpenseTrackerController.java` to support `applyFilter()`.
- Update the `ExpenseTrackerView.java` to allow filtering by *amount* or *category*.
- Apply Homework 1's input validation on filter input fields.

This new feature should be **committed** to your git repository.

Testability [20 points]

- Each test case needs to apply the commonly used test case template shown in the unit testing lecture.
- Ensure existing 2 test cases still pass.
- Add the following 4 new test cases:
 1. Add Transaction:
 - Steps: For example, add a transaction with amount 50.00 and category "food"
 - Expected Output: Transaction is added to the table, Total Cost is updated
 2. Invalid Input Handling:
 - Steps: Attempt to add a transaction with an invalid amount or category
 - Expected Output: Error messages are displayed, transactions and Total Cost remain unchanged
 3. Filter by Amount:
 - Steps: Add multiple transactions with different amounts, apply amount filter
 - Expected Output: Only transactions matching the amount are returned (and will be displayed)
 4. Filter by Category:
 - Steps: Add multiple transactions with different categories, apply category filter
 - Expected Output: Only transactions matching the category are returned (and will be displayed)
- Place these new tests in `test/` folder and **commit** them.
- Ensure all new test cases pass.
- Include a screenshot from your IDE showing the test runner with all the tests passing.

Usability: Undo Functionality [20 points]

One good UI design principle is to provide undo functionality. Here is a basic idea for implementing undo for this app:

- User can remove a transaction by selecting the row.
- The removal updates the `Total Cost`.

The undo design should follow MVC. You are responsible for providing the necessary changes to the Model, View and Controller. You do not need to implement this section, submit it as a written plan in a plain-text document `undo.txt` or PDF document `undo.pdf`.

Deliverables [Approximately 16 points]

1. Design document for undo (`undo.txt` or `undo.pdf`)
2. Generated javadoc in a folder `jdoc/`
3. Git log (`gitlog.txt` or `gitlog.pdf`) with meaningful commits
4. Screenshot of test runner (`test_screenshot.png`)
5. Application compiles and runs successfully
6. Test suite compiles, runs, and all test cases pass

Submission Format

Submit a folder or zip file named: hw2_expensetracker/

It must contain the following structure:

```
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  # Git commit log  
  test_screenshot.png # Test runner screenshot  
  undo.txt or .pdf    # Undo design document
```

Do not include the .git folder. Submit via [Gradescope](#).