Capstone Planning Guide

Project Overview

You are building a console application to track financial transactions for a business or personal use. Your application must load transactions from a CSV file, allow users to add deposits and payments, view a ledger of transactions, and optionally generate reports.

Core Requirements (Passing)

- Home Screen with options:
  - Add Deposit
  - Make Payment
  - View Ledger
  - Exit
- Add Deposit: Save a new deposit to the CSV file. Deposits must have a positive amount.
- Make Payment: Save a new payment to the CSV file. Payments must have a negative amount (even if the user enters a positive number).
- View Ledger: Display all transactions, newest first, with options to filter by deposits or payments.
- Application continues running until the user chooses Exit.

Bonus Features (Exceeding)

- Reports Screen (2 reports minimum):
  - Month-To-Date
  - Previous Month
  - Year-To-Date

    - Previous Year
    - Search by Vendor
- Optional Challenge: Custom Search

My biggest piece of advice for this project is to look at previous examples of how to do things and rework them for the current problem.

Suggested Class Structures

There are different ways to organize your code to accomplish this project. We'll start with the easiest and move up to more complicated. Pick which level of organization makes sense to your brain and go with that.

Basic Structure (Quick but Messy)

- Main: Handles everything (menus, displaying transactions, filtering transactions, all user input).
- Transaction: Represents a single transaction.
- TransactionFileManager: Loads and saves transactions, works with the .csv file

Pros: Easier to build quickly.
Cons: Main becomes very large and hard to manage.

Intermediate Structure (Cleaner)

- Main: Handles menu navigation and major decision-making.
- Transaction: Represents a transaction.
- TransactionFileManager: Handles file operations.
- Ledger: Holds the list of transactions and handles filtering, searching, and reports. These methods are static. No getters/setters, constructor loads transactions from file.

Pros: Better separation of concerns. Easier to find and fix code later.
Cons: Slightly more setup required.

Advanced Structure (More Professional Level)

- Main: Very thin. Only starts the program and controls flow.

- UserInterface: Handles displaying menus, getting user input, and calling methods on other classes.
- Ledger: Manages transactions (filtering, searching, managing the list).
- TransactionFileManager: Loads and saves transaction data to and from CSV.
- Transaction: Represents a single transaction.

How classes interact:

- Main talks to UserInterface
- UserInterface talks to Ledger
- Ledger talks to TransactionFileManager
- TransactionFileManager reads/writes transactions
- Transaction is just a data object

Pros: Very clean, scalable design. Each class has a clear job.
Cons: Takes a little longer to set up, but easier to add new features later.

You can do this project with just a Main Class, a FileManager class, and a Transaction class. You can use methods in the main class to break up the logic similar to Workshop #3. The other structures are if you feel comfortable with the basic structure and want to challenge yourself to make your project more realistic.

Development Order (Recommended)

1. Create the Transaction class

- Represents a single transaction.
- Define all necessary properties, constructor, and getters/setters.
- This class is a POJO (Plain ol' Java Object). It doesn't do anything special except hold data.

2. Create the TransactionFileManager class

- Responsible for reading from and writing to the CSV file.
- Helps separate file handling from your main application logic.
- NOT a POJO, meaning it does something special.

3. Set up the Main class and Home Screen

- Build a basic menu that can load dummy data and display options.

4. Implement Add Deposit and Make Payment

- Collect user input.
- Create a new Transaction.
- Save the new Transaction to the CSV file.

5. Implement the Ledger screen

- Display all transactions by reading them from the file.
- Add filtering for deposits and payments.

6. Build the Reports screen (if time allows)

- Provide Month-To-Date, Previous Month, Year-To-Date, Previous Year, and Vendor Search options.

Java Features and Tools You Will Likely Use

- LocalDate and LocalTime from java.time
- DateTimeFormatter for formatting and parsing
- BufferedReader and BufferedWriter for CSV file handling
- ArrayList for storing transactions
- String splitting (String.split("\\|")) to process CSV lines

---

Final Tips

- Start small and test each part before moving on.
- Use hardcoded examples to test functionality before fully wiring up file reading or writing.
- Stub out methods if you are unsure how to write them yet.
- Focus on finishing the core functionality first, then polish.

Planning Worksheet

Before you start coding, answer these questions:

1. What properties will your Transaction class have?
2. What methods will your TransactionFileManager class need?
3. If you create a Ledger class, what methods should it have?
4. How will the Home Screen be organized? What options will it show?
5. Write out your menus by hand! Be clear on how the user moves from menu to menu.
6. What information will you need to prompt the user for when adding a new transaction?
7. How will you decide if an amount should be positive or negative?
8. What filtering options will you offer for the Ledger screen?
9. How will you filter? Are there any examples from previous workshops to offer a guide?
10. What class will be responsible for reading from and writing to the CSV file?

The biggest most helpful question is “Have I done something similar to this before and if so, how did I accomplish it?”