Brenda Romero Torres
CPSC-39-12705
Personal Finance Budget App

Overview of My App (what it does & why it's useful)

For my final project, I created a Personal Finance Budget App in Java. The goal of this app is to help users keep better track of their income, expenses, and overall spending habits. The user can add transactions, view their full history, check detailed summaries, undo mistakes, and even schedule future bills that get processed automatically later.

I wanted this project to be something modern and actually useful, especially for college students who want to watch their spending. The app organizes everything into clean sections, and it also saves data so nothing is lost when the user closes it. I added color formatting and a cleaner menu system to make the experience feel a little more like a real budgeting tool, even though it runs in the terminal.

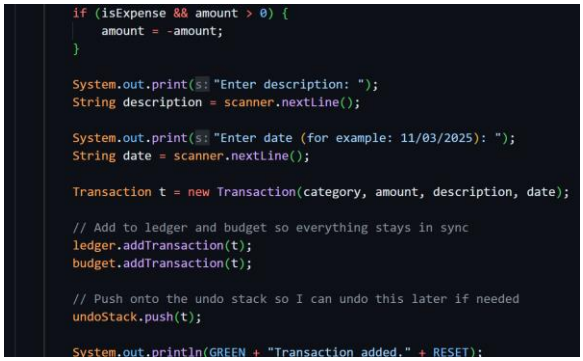
Algorithms

Algorithm 1: Add Transaction

Purpose:

Take user input, create a transaction, and update everything that needs to be updated in the app.

Steps:



1. Ask the user for a category.
2. Ask if the transaction is Income or Expense.
3. Ask for the amount as a positive number.
4. IF the transaction is an Expense THEN
   - Turn the amount into a negative number
5. ENDIF
6. Ask for the description.
7. Ask for the date.
8. Create a new Transaction object.
9. Add it to the Ledger (LinkedList).
10. Update the Budget totals and category totals (HashMap).
11. Push the transaction onto the Undo Stack.
12. Print a confirmation.

Algorithm 2: Recursive Total Expense Calculation

Purpose:

Use recursion to walk through the transaction list and add up all expenses.

Steps:

1. IF index equals the size of the list THEN
   - RETURN 0
2. ENDIF
3. Read the transaction at the current index.
4. IF the amount is negative THEN
   4.1 Set expense = absolute value of the amount
   ELSE
   4.2 Set expense = 0
5. ENDIF
6. RETURN expense + RecursiveCall(list, index + 1)

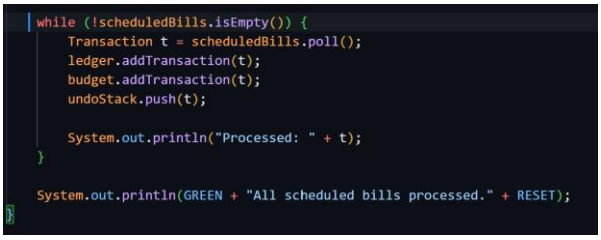
Algorithm 3: Process Scheduled Bills

Purpose:

Process bills the user scheduled earlier using FIFO order.

Steps:

1. WHILE the scheduledBills queue is not empty DO
   - Remove the first scheduled bill
   - Add it to the Ledger
   - Update the Budget totals
   - Add it to the Undo Stack
   - Print a confirmation message
2. ENDWHILE
3. Print “All scheduled bills processed.”



How I Created These Algorithms

Add Transaction:

This is one of the main algorithms in the app, and I built it to handle everything that happens when the user adds something new. I broke it into simple steps so the rest of the app stays organized. I came up with the logic on my own based on what I wanted the app to do. ChatGPT mainly helped me keep things structured and made sure I didn’t miss any important steps.

Recursive Expense Calculation:

I made this algorithm to meet the recursion requirement, but I also wanted it to actually be helpful in the app. I designed the logic to walk through each transaction and only add up the expenses. ChatGPT helped explain how to structure recursion properly, especially with the base case, but I personally decided exactly what it should calculate and how it should work.

Process Scheduled Bills:

This algorithm uses a queue because scheduled bills should always be handled in the order they were added. I wrote the general flow myself, and ChatGPT helped me check that FIFO logic made sense for this type of feature. This ended up being a pretty realistic feature that made the app feel more complete.

Big-O Time for Each Algorithm

Add Transaction, O(1), Everything is constant time (adding to list, updating HashMap, stack push).

Recursive Total Expense Calculation, O(n), The recursion visits each transaction once.

Process Scheduled Bills, O(n), Each bill is processed once from the queue.

Data Structures I Used and Why

LinkedList: I used a LinkedList to store all the transactions because it's simple to work with and keeps them in the exact order they were added.

HashMap: The HashMap stores category totals like “Food,” “Bills,” etc. It’s fast, and updating totals is super easy.

Stack: I used a stack for the undo feature since undoing something always works in reverse order.

Queue: Scheduled bills go into a queue because bills should be processed in the order they’re scheduled.

Arrays: I used arrays to store monthly income and expense totals. Each index represents a month, which made the math straightforward.

Opportunity & Bug I Saw While Working on the App

While building the app, I noticed that the constant menu printing made everything feel messy. So I took the opportunity to improve the user experience by cleaning up the layout, adding colors, and showing the menu only when the user asks for it. These small changes made the app feel a lot more organized and actually nicer to use.

I had a bug where scheduled bills were showing as income instead of expenses. The issue was that I wasn’t automatically turning the amount into a negative number. Once I fixed that and forced all scheduled bill amounts to be negative, everything worked the way it was supposed to.

What I Would Add & Update

If I continued this project, I'd want to add:

- A nice graphical interface
- Charts to visually show spending habits
- Exportable reports (like a PDF)
- User login system
- Better search features for filtering transactions

These changes would make the app feel like a real budget tool people could actually rely on.

Personal Finance Budget App — Project Summary

My project is a Personal Finance Budget App that helps users track their income and expenses. It uses multiple classes, data structures, and algorithms to manage transactions, generate summaries, and let users undo actions or process scheduled bills.

I used a LinkedList to store the ledger, a HashMap for category totals, arrays for monthly summaries, a Stack for undo, and a Queue for scheduled bills. I also created a recursive function to calculate total expenses. This project meets all SLO requirements because it uses all the required data structures, includes recursion, uses object-oriented design, and demonstrates abstraction and memory management in Java.

My report includes three algorithms written in the required format, snapshots of the code, Big-O time analysis, explanations of the data structures, and reflections on an opportunity I had during development (improving the interface) and a bug I fixed (scheduled bills being treated as income). For future versions, I would add a GUI, charts, exporting tools, and more search features.