Billify

Anthony Clemens

St. Petersburg College

COP 4504

Professor V

5/6/2024

Table of Contents

Description of Software

Billify will be a standalone piece of software designed to help users manage their bills, expenses, and paychecks easily. It aims to provide an all-in-one way for users to track their financial transactions, net balances, and take a look into their spending patterns.

Perspective

Billify will be a standalone app that is not part of a larger project, as it will have all of the desired features built in. The end user will interface with Billify through a GUI that will provide all of the graphs and data entry features.

Functions

Billify will include expense tracking, allowing users to input and categorize their expenses, which includes their bills, groceries, insurances, entertainment, and savings, etc. Income management will also be included in Billify via the ability to add paychecks, freelancing deposits for things such as Doordash or Instacart, and other forms of income. Another function would be calculating how much money is left after a selectable timeframe, as low as a day, and as large as a year, allowing the user to get a wide range of information on how much money is left over or missing after expenses. In addition to net balance calculation, the end-user can view summaries, charts, and trends all related to their financial activities, giving the user total control over how they can view their expenses and income.

Environment

Billify will be a standalone Java app that can run on any platform. The only thing that will need to be installed is a java environment on the host system, and Java can run on almost anything, Windows, Mac OS, and Linux. This is due to Java being a “WORA” language, Write Once, Run Anywhere, making Billify a multiplatform application with little to no code changes over different platforms. Billify will implement a Java graphical library JavaFX to create a GUI that is responsive and multi-platform friendly.

Requirements

Functional Requirements

Feature 1: Expenses

Expense tracking with categorization, such as rent, utilities, phone bill, rent, savings, etc.

Feature 2: Income

Income tracking, allowing addition of paychecks, bonuses, freelancing income, etc.

Feature 3: Net Balance

Net balance calculation, showing income minus expenses, allowing users to see how much is left over.

Use Case 1:

A user adds a new expense or income to the data set.

Use Case 2:

An end user views a summary of their expenses and income.

Non Functional Requirements

Performance:

The application needs to be very responsive, and low data retrieval times and low calculation times.

Useability:

The user interface needs to be good looking and intuitive for even novice users. If a more advanced experience is desired, advanced options can be available via menus.

Portability:

Data will be stored in .CSV files, which allows for amazing portability since it is lightweight and compatible with other programs such as Microsoft Excel.

Constraints

The program will be written in Java, using the JavaFX library to make a responsive GUI, and the ability to export and import .CSV files for simplicity. It will have to be compatible with all operating systems.

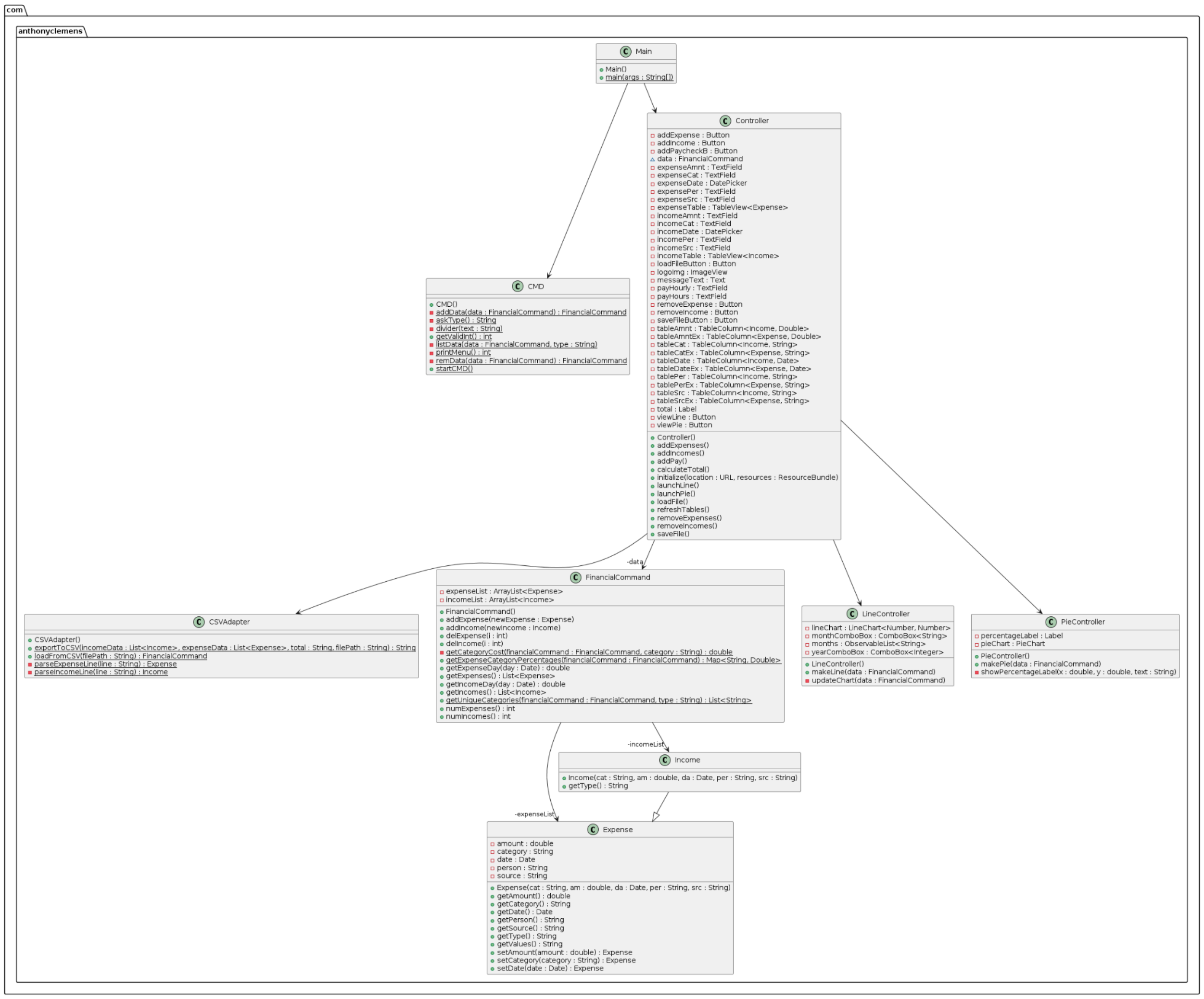
User Interface Requirements

The user interface will have to be easy to use for novice users, while having the functionality available to more advanced users via extra menus, or maybe an "advanced" option in the settings. Visual feedback will be ever present in the program, such as pie charts, line charts, etc. The program will also have to be scalable to different screen sizes, whether it be big or small.

Other Requirements

Provide the ability to make backups on data, or autosave data entries in the event of a crash or power loss. Another nice addition would be to allow the user to export the data, via .png's of the graphs, .CSV files for the data entries, etc.

UML Diagram


com
anthonyclemens
Main
Main()
main(args : String[])
Controller
addExpense : Button
addIncome : Button
addPaycheckB : Button
data : FinancialCommand
expenseAmnt : TextField
expenseCat : TextField
expenseDate : DatePicker
expensePer : TextField
expenseSrc : TextField
expenseTable : TableView<Expense>
incomeAmnt : TextField
incomeCat : TextField
incomeDate : DatePicker
incomePer : TextField
incomeSrc : TextField
incomeTable : TableView<Income>
loadFileButton : Button
logoImg : ImageView
messageText : Text
payHourly : TextField
payHours : TextField
removeExpense : Button
removeIncome : Button
saveFileButton : Button
tableAmnt : TableColumn<Income, Double>
tableAmntEx : TableColumn<Expense, Double>
tableCat : TableColumn<Income, String>
tableCatEx : TableColumn<Expense, String>
tableDate : TableColumn<Income, Date>
tableDateEx : TableColumn<Expense, Date>
tablePer : TableColumn<Income, String>
tablePerEx : TableColumn<Expense, String>
tableSrc : TableColumn<Income, String>
tableSrcEx : TableColumn<Expense, String>
total : Label
viewLine : Button
viewPie : Button
Controller()
addExpenses()
addIncomes()
addPay()
calculateTotal()
initialize(location : URL, resources : ResourceBundle)
launchLine()
launchPie()
loadFile()
refreshTables()
removeExpenses()
removeIncomes()
saveFile()
CMD
CMD()
addData(data : FinancialCommand) : FinancialCommand
askType() : String
divider(text : String)
getValidInt() : int
listData(data : FinancialCommand, type : String)
printMenu() : int
remData(data : FinancialCommand) : FinancialCommand
startCMD()
-data
CSVAdapter
CSVAdapter()
exportToCSV(incomeData : List<Income>, expenseData : List<Expense>, total : String, filePath : String) : String
loadFromCSV(filePath : String) : FinancialCommand
parseExpenseLine(line : String) : Expense
parseIncomeLine(line : String) : Income
FinancialCommand
expenseList : ArrayList<Expense>
incomeList : ArrayList<Income>
FinancialCommand()
addExpense(newExpense : Expense)
addIncome(newIncome : Income)
delExpense(i : int)
delIncome(i : int)
getCategoryCost(financialCommand : FinancialCommand, category : String) : double
getExpenseCategoryPercentages(financialCommand : FinancialCommand) : Map<String, Double>
getExpenseDay(day : Date) : double
getExpenses() : List<Expense>
getIncomeDay(day : Date) : double
getIncomes() : List<Income>
getUniqueCategories(financialCommand : FinancialCommand, type : String) : List<String>
numExpenses() : int
numIncomes() : int
LineController
lineChart : LineChart<Number, Number>
monthComboBox : ComboBox<String>
months : ObservableList<String>
yearComboBox : ComboBox<Integer>
LineController()
makeLine(data : FinancialCommand)
updateChart(data : FinancialCommand)
PieController
percentageLabel : Label
pieChart : PieChart
PieController()
makePie(data : FinancialCommand)
showPercentageLabel(x : double, y : double, text : String)
-incomeList
Income
Income(cat : String, am : double, da : Date, per : String, src : String)
getType() : String
-expenseList
Expense
amount : double
category : String
date : Date
person : String
source : String
Expense(cat : String, am : double, da : Date, per : String, src : String)
getAmount() : double
getCategory() : String
getDate() : Date
getPerson() : String
getSource() : String
getType() : String
getValues() : String
setAmount(amount : double) : Expense
setCategory(category : String) : Expense
setDate(date : Date) : Expense

The Main class starts with the program launch, giving the option to the command line simple interface, or the JavaFX implementation of the program. The command line version is used for testing out the FinancialCommand class, the main meat of the whole program. The FinancialCommand class creates an ArrayList of Expense and Income objects, of which have a date, amount, person, source, and category. All math relating to the ArrayList of Income and Expense objects are done in FinancialCommand. Controller is the JavaFX class that runs the GUI, which holds a FinancialCommand object, and can call on the PieChart class, the LineChart Graph, or the CSVAdapter to either create a pie chart, create a line chart, or save/load a CSV file.

Design Pattern(s)

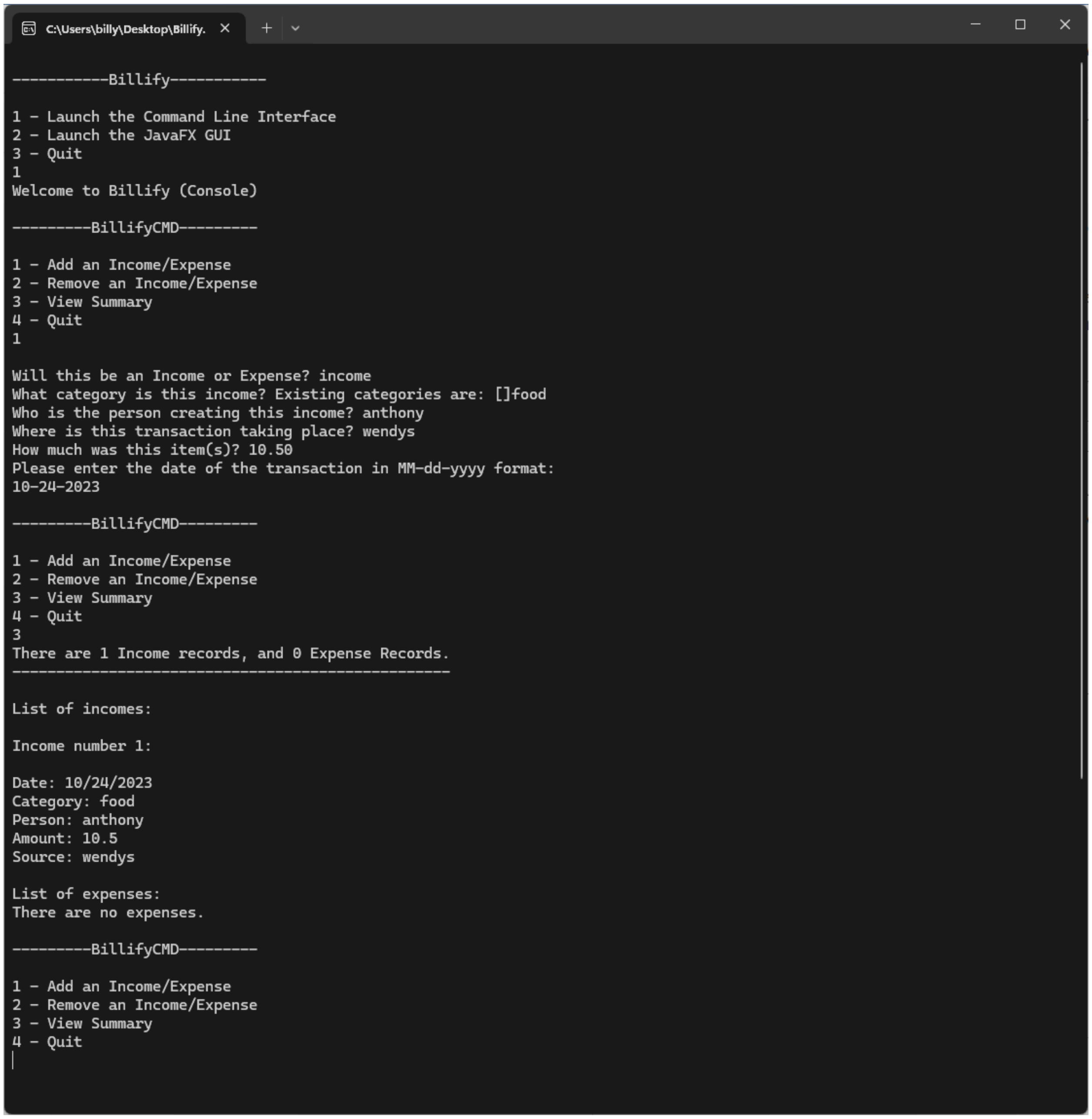
For Billify, I actually used multiple different design patterns. The Controller class is the central code for the GUI backend, and is typical of an MVC pattern. The FinancialCommand class would be considered the model, since it contains the data for the application and the business logic. The view is represented by the UI elements generated and put to the display via the Controller, with buttons, text fields, and date pickers. Billify also uses the Adapter pattern, due to the CSVAdapter class, where I convert the ArrayLists of Income and Expense objects, of which all I have to do is pass on existing code to the Adapter without changing the core business logic.

Releases with Screenshots

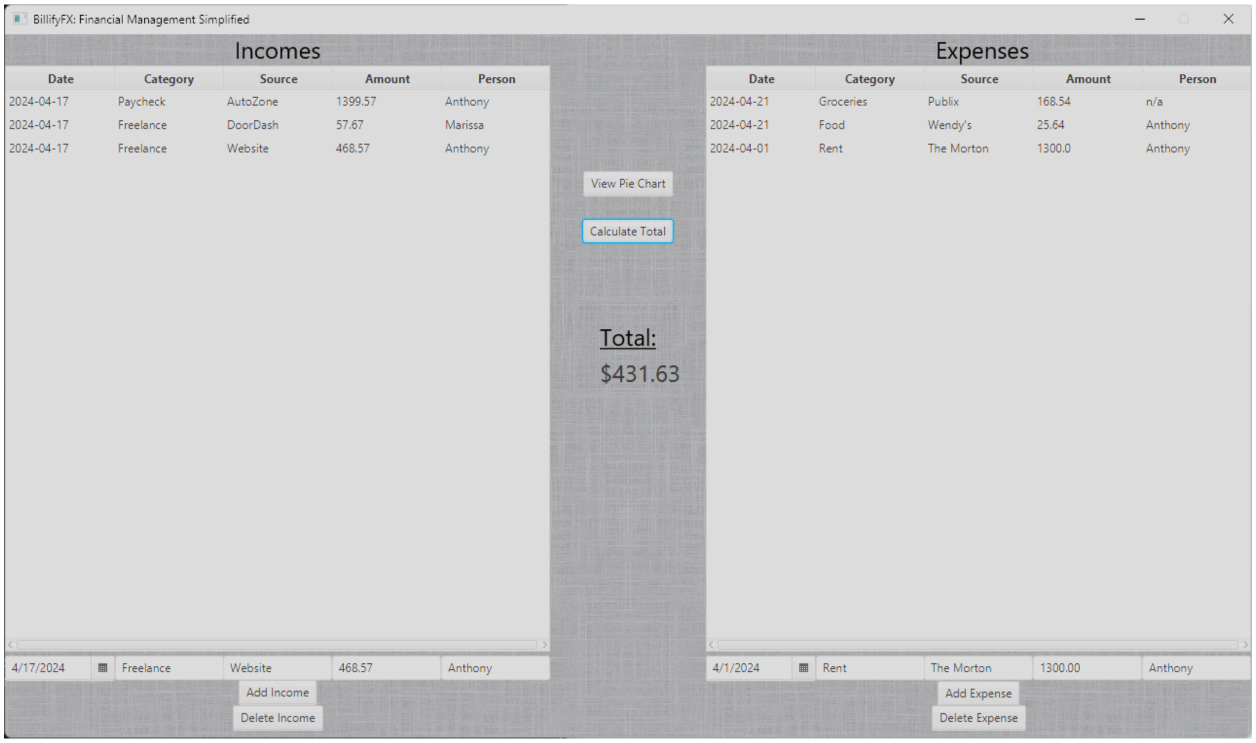
First Release (v0.3):

https://github.com/AnthonyClemens/Billify/tree/v0.3

This release had the Command Line version, and a simple GUI for adding and subtracting Incomes and Expenses, which would only update the total when “Calculate Total” was pressed. This version had many bugs, and lack of input checking, so many errors would show in the console, sometimes causing crashes. This version also had a lot of redundant code and argument passing.



```
-----------Billify-----------

1 - Launch the Command Line Interface
2 - Launch the JavaFX GUI
3 - Quit
1
Welcome to Billify (Console)

---------BillifyCMD---------

1 - Add an Income/Expense
2 - Remove an Income/Expense
3 - View Summary
4 - Quit
1

Will this be an Income or Expense? income
What category is this income? Existing categories are: []food
Who is the person creating this income? anthony
Where is this transaction taking place? wendys
How much was this item(s)? 10.50
Please enter the date of the transaction in MM-dd-yyyy format:
10-24-2023

---------BillifyCMD---------

1 - Add an Income/Expense
2 - Remove an Income/Expense
3 - View Summary
4 - Quit
3
There are 1 Income records, and 0 Expense Records.
--------------------------------------------------

List of incomes:

Income number 1:

Date: 10/24/2023
Category: food
Person: anthony
Amount: 10.5
Source: wendys

List of expenses:
There are no expenses.

---------BillifyCMD---------

1 - Add an Income/Expense
2 - Remove an Income/Expense
3 - View Summary
4 - Quit
```



Incomes

Date	Category	Source	Amount	Person
2024-04-17	Paycheck	AutoZone	1399.57	Anthony
2024-04-17	Freelance	DoorDash	57.67	Marissa
2024-04-17	Freelance	Website	468.57	Anthony

4/17/2024 Freelance Website 468.57 Anthony

Expenses

Date	Category	Source	Amount	Person
2024-04-21	Groceries	Publix	168.54	n/a
2024-04-21	Food	Wendy's	25.64	Anthony
2024-04-01	Rent	The Morton	1300.0	Anthony

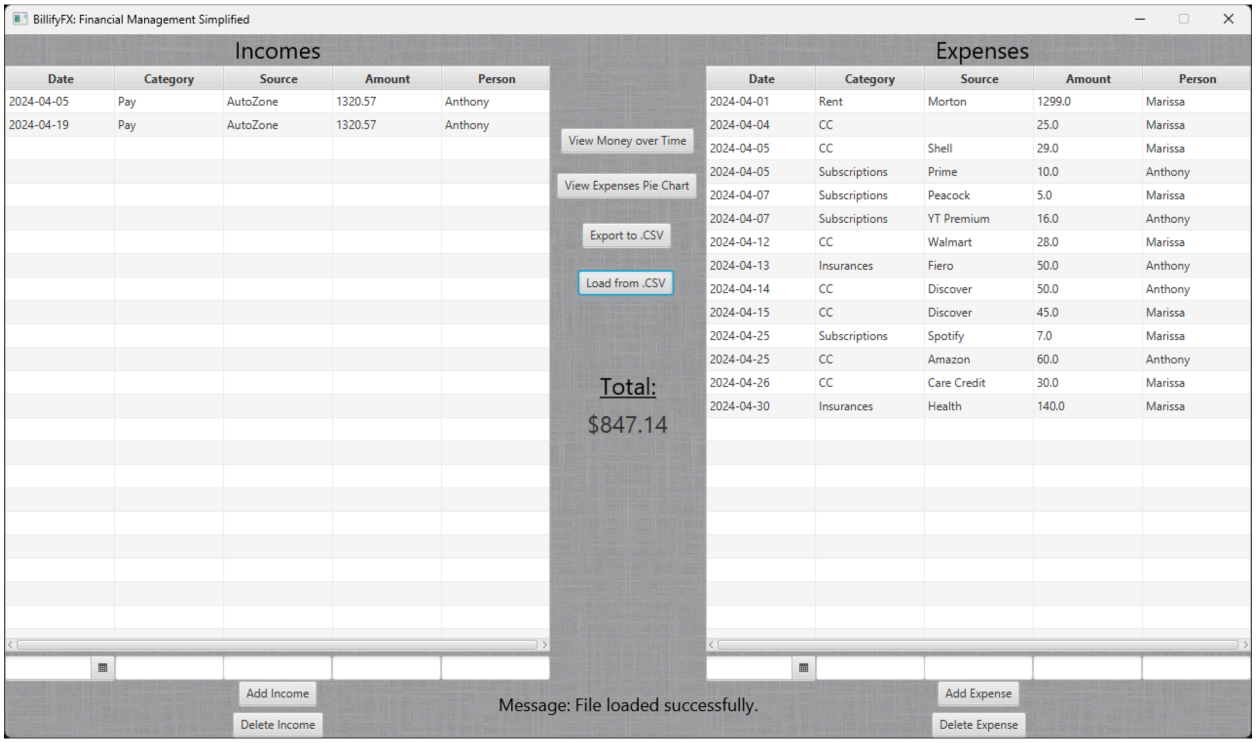
4/1/2024 Rent The Morton 1300.00 Anthony

Total:
$431.63

Second Release (v0.4):

https://github.com/AnthonyClemens/Billify/tree/v0.4

The second release I enjoyed a lot more due to the addition of the Line Chart of money over time per day of the month, and the addition of the Pie Chart showing percentages when you hover over it. CSV saving and loading was also added in this version, which moved the program up into the usable category. Instead of crashing or having errors put to the console, I made a Message bar at the bottom, of which I could catch Exceptions, and post relevant messages to the user on how to fix their problem, and not crash the program.



Incomes

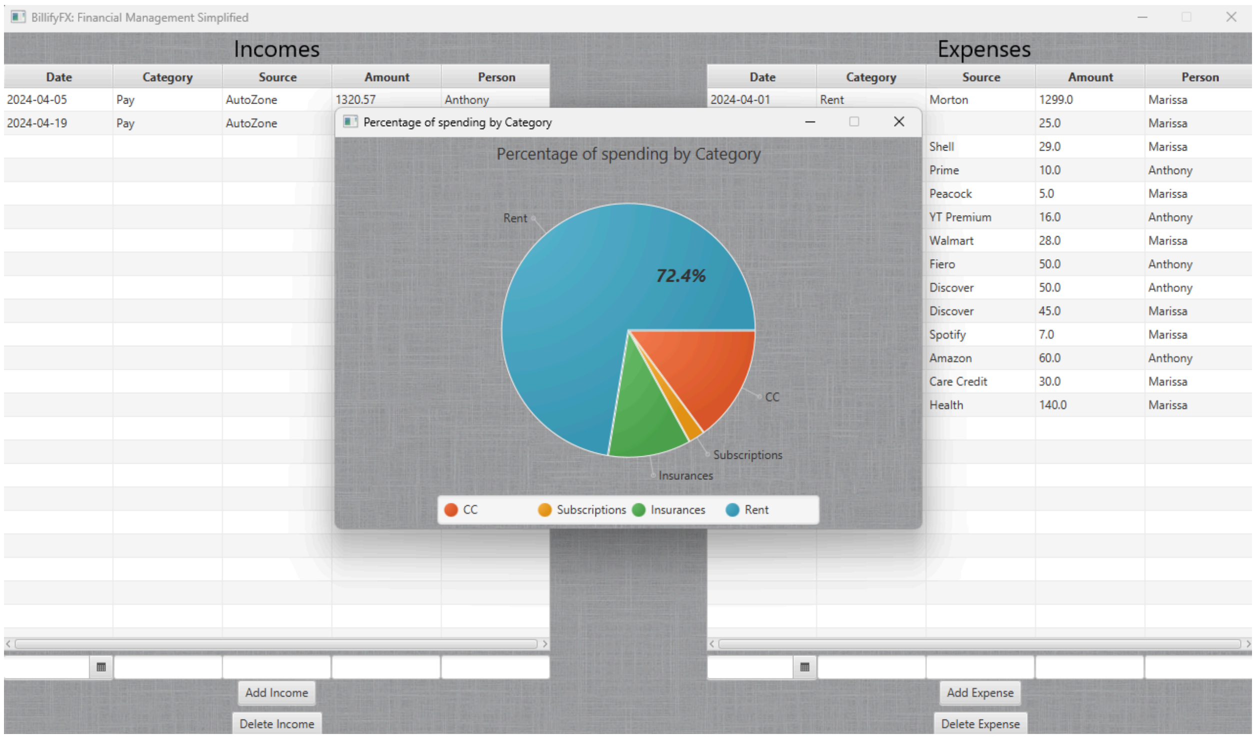
Date	Category	Source	Amount	Person
2024-04-05	Pay	AutoZone	1320.57	Anthony
2024-04-19	Pay	AutoZone	1320.57	Anthony

Expenses

Date	Category	Source	Amount	Person
2024-04-01	Rent	Morton	1299.0	Marissa
2024-04-04	CC		25.0	Marissa
2024-04-05	CC	Shell	29.0	Marissa
2024-04-05	Subscriptions	Prime	10.0	Anthony
2024-04-07	Subscriptions	Peacock	5.0	Marissa
2024-04-07	Subscriptions	YT Premium	16.0	Anthony
2024-04-12	CC	Walmart	28.0	Marissa
2024-04-13	Insurances	Fiero	50.0	Anthony
2024-04-14	CC	Discover	50.0	Anthony
2024-04-15	CC	Discover	45.0	Marissa
2024-04-25	Subscriptions	Spotify	7.0	Marissa
2024-04-25	CC	Amazon	60.0	Anthony
2024-04-26	CC	Care Credit	30.0	Marissa
2024-04-30	Insurances	Health	140.0	Marissa

Total:
$847.14

Message: File loaded successfully.

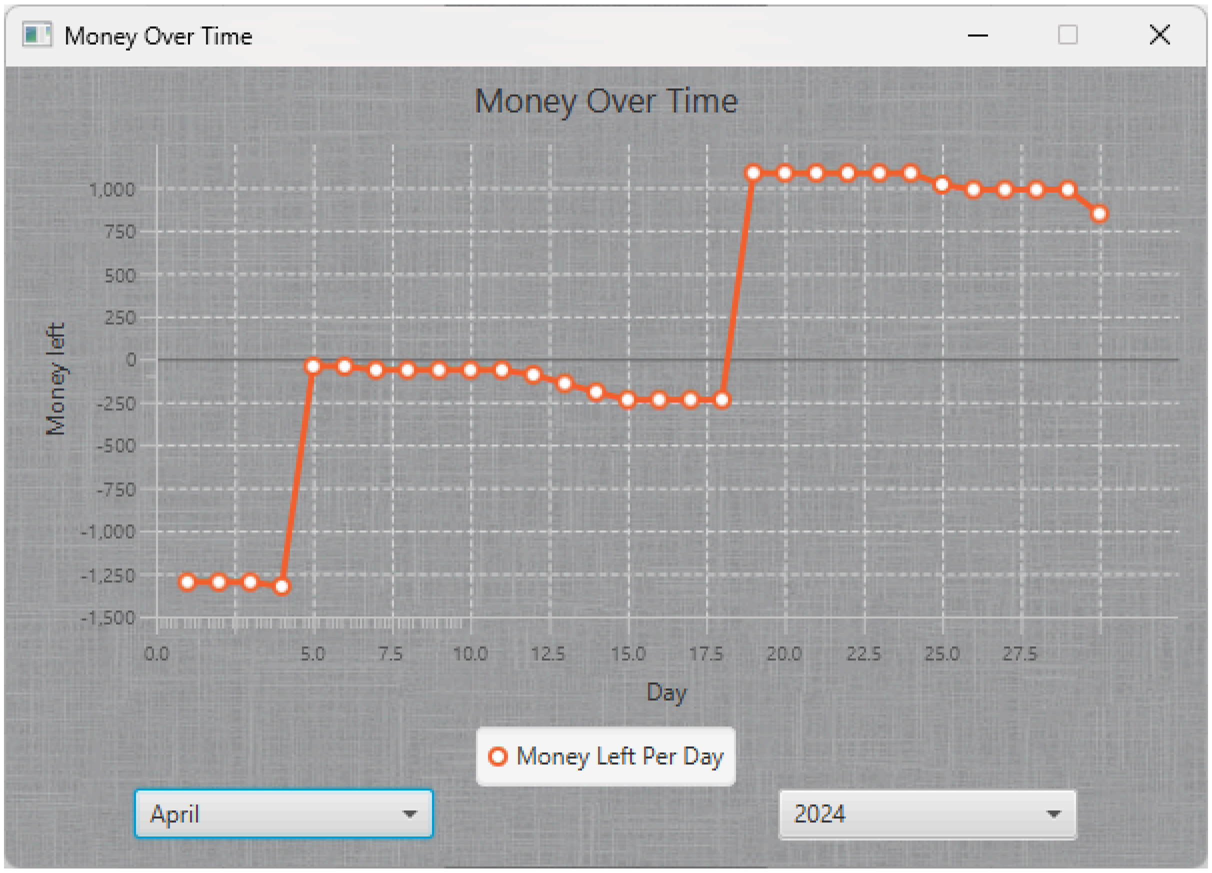

BillifyFX: Financial Management Simplified
Percentage of spending by Category
Percentage of spending by Category
Rent
72.4%
CC
Subscriptions
Insurances
● CC ● Subscriptions ● Insurances ● Rent
Add Income
Delete Income
Add Expense
Delete Expense

Incomes

Date	Category	Source	Amount	Person
2024-04-05	Pay	AutoZone	1320.57	Anthony
2024-04-19	Pay	AutoZone		

Expenses

Date	Category	Source	Amount	Person
2024-04-01	Rent	Morton	1299.0	Marissa
			25.0	Marissa
		Shell	29.0	Marissa
		Prime	10.0	Anthony
		Peacock	5.0	Marissa
		YT Premium	16.0	Anthony
		Walmart	28.0	Marissa
		Fiero	50.0	Anthony
		Discover	50.0	Anthony
		Discover	45.0	Marissa
		Spotify	7.0	Marissa
		Amazon	60.0	Anthony
		Care Credit	30.0	Marissa
		Health	140.0	Marissa

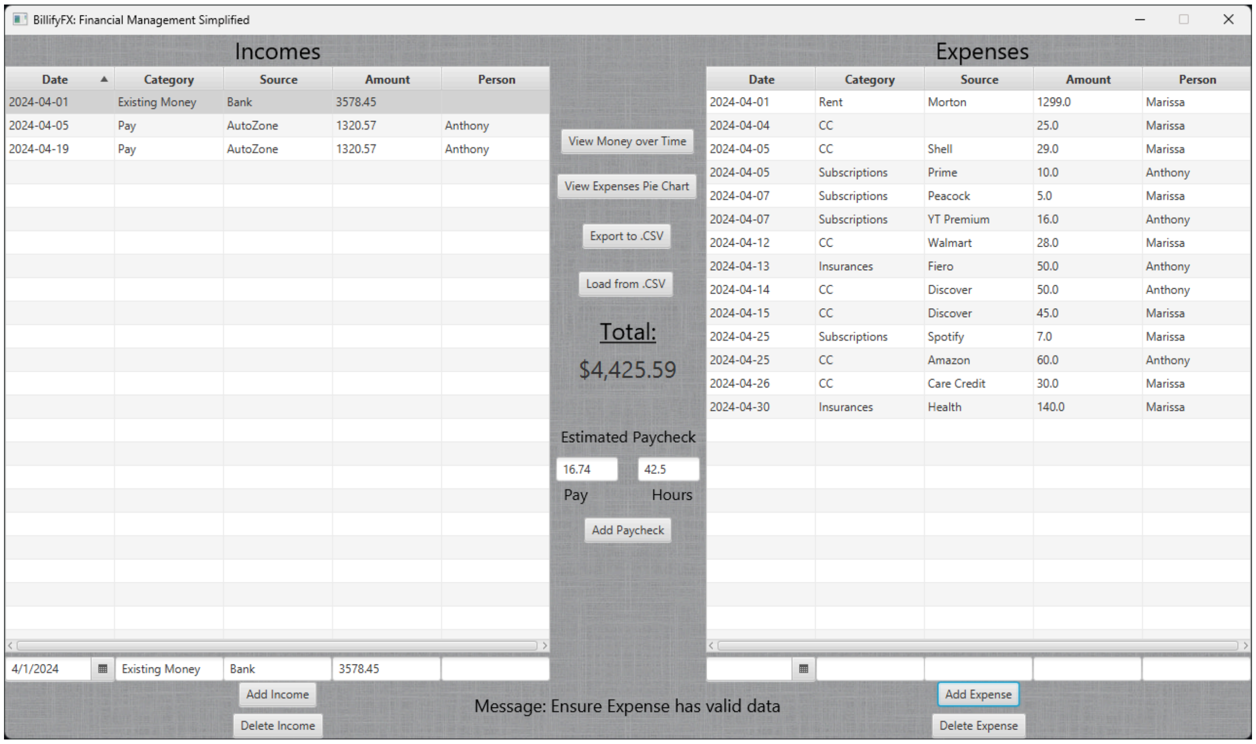


Final Release (v1.0):

https://github.com/AnthonyClemens/Billify/tree/v1.0

For the Final release, I took out the Command Line version as it was very lacking in features, and was mainly to test the functionality of the FinancialCommand class. I took out a ton of redundant code, and redundant arguments being passed through methods. I also added a paycheck calculator, that even goes based off of tax bracket and overtime calculations depending on if certain requirements are met. This is the most polished and bug-free version of my program, and I would say this is a good stopping point. It has the features I would want for myself to keep track of my bills. Yes it is not perfect, but it was my first Maven project, my first big JavaFX project, and my first multi-file project that I have done. If I had done this again I would have laid out certain classes differently in order to enable more functionality from

JavaFX, since the way I called new windows was apparently not the best solution, locking me out of features I wanted to use.



Incomes

Date	Category	Source	Amount	Person
2024-04-01	Existing Money	Bank	3578.45	
2024-04-05	Pay	AutoZone	1320.57	Anthony
2024-04-19	Pay	AutoZone	1320.57	Anthony
4/1/2024	Existing Money	Bank	3578.45	

Total:
$4,425.59

Estimated Paycheck
Pay: 16.74
Hours: 42.5

Expenses

Date	Category	Source	Amount	Person
2024-04-01	Rent	Morton	1299.0	Marissa
2024-04-04	CC		25.0	Marissa
2024-04-05	CC	Shell	29.0	Marissa
2024-04-05	Subscriptions	Prime	10.0	Anthony
2024-04-07	Subscriptions	Peacock	5.0	Marissa
2024-04-07	Subscriptions	YT Premium	16.0	Anthony
2024-04-12	CC	Walmart	28.0	Marissa
2024-04-13	Insurances	Fiero	50.0	Anthony
2024-04-14	CC	Discover	50.0	Anthony
2024-04-15	CC	Discover	45.0	Marissa
2024-04-25	Subscriptions	Spotify	7.0	Marissa
2024-04-25	CC	Amazon	60.0	Anthony
2024-04-26	CC	Care Credit	30.0	Marissa
2024-04-30	Insurances	Health	140.0	Marissa

Message: Ensure Expense has valid data