Project Proposal

Project Summary

Proposed Project Title: Money Buddy (Personal Financial Tracker)

Money Buddy is a personal financial tracker application that helps users effectively manage their income, expenses, and savings goals. The application provides a simple command-line interface for users to input and keep track of their financial transactions for a specific month and year, set and review savings goals, and analyze their financial performance through reports.

Intended User

The intended user is anyone looking to manage their personal finances, track income and expenses, and set savings goals to better plan their financial future.

What problem is the project trying to solve?

The project aims to address the challenge of effectively managing personal finances by providing users with an easy-to-use tool that consolidates financial information, allowing them to track income and expenses, set savings goals, and analyze their financial performance. Furthermore, if the user feels unsafe connecting their bank account to a third party app to use these tools, they can simply resort to Money Buddy

Which technologies will you need?

- Java Development Kit (JDK) to run an executable JAR file
- A serialized file ('financial_data.ser') for storing and loading user data.
- A CSV file export for generating financial reports.
- A command-line interface (CLI) for user interaction.

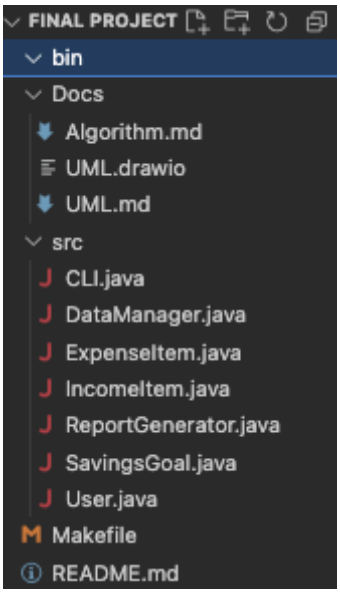
Use Case Analysis

There are two ways you can start MoneyBuddy:

Using an executable jar file

1. Make sure there is nothing in `bin` directory otherwise run `make clean` on the terminal

```
➜  Final Project git:(master) make clean
rm -f bin/*.class
➜  Final Project git:(master)
```



2. run `make jar` on terminal

```
➜  Final Project git:(master) ✗ make jar
javac -d bin -sourcepath src src/CLI.java
javac -d bin -sourcepath src src/DataManager.java
javac -d bin -sourcepath src src/ExpenseItem.java
javac -d bin -sourcepath src src/IncomeItem.java
javac -d bin -sourcepath src src/ReportGenerator.java
javac -d bin -sourcepath src src/SavingsGoal.java
javac -d bin -sourcepath src src/User.java
echo 'Main-Class: CLI' > manifest.txt
jar cvfm MoneyBuddy.jar manifest.txt -C bin .
added manifest
adding: CLI.class(in = 8522) (out= 4244)(deflated 50%)
adding: DataManager.class(in = 1320) (out= 743)(deflated 43%)
adding: ExpenseItem.class(in = 1074) (out= 518)(deflated 51%)
adding: IncomeItem.class(in = 882) (out= 454)(deflated 48%)
adding: ReportGenerator.class(in = 3586) (out= 1811)(deflated 49%)
adding: SavingsGoal.class(in = 1623) (out= 725)(deflated 55%)
adding: User.class(in = 6026) (out= 2762)(deflated 54%)
rm -f manifest.txt
➜  Final Project git:(master) ✗
```

3. Now, make sure you see the `executable jar` file in the project directory

4. Now run `java -jar MoneyBuddy.jar` on the terminal, which will run the program

```
→  Final Project git:(master) ✗ java -jar MoneyBuddy.jar
Error loading user data: financial_data.ser (No such file or directory)
Welcome to the Financial Tracker!

Money Buddy Main Menu:
========================
1. Manage Income
2. Manage Expenses
3. Set Savings Goal
4. View Financial Summary
5. Export Data
6. Exit
Enter your choice:
```

Using make and make run

1. Run `make` and `make run` on the terminal

```
→  Final Project git:(master) ✗ make
javac -d bin -sourcepath src src/CLI.java
javac -d bin -sourcepath src src/DataManager.java
javac -d bin -sourcepath src src/ExpenseItem.java
javac -d bin -sourcepath src src/IncomeItem.java
javac -d bin -sourcepath src src/ReportGenerator.java
javac -d bin -sourcepath src src/SavingsGoal.java
javac -d bin -sourcepath src src/User.java
→  Final Project git:(master) ✗ make run
javac -d bin -sourcepath src src/CLI.java
javac -d bin -sourcepath src src/DataManager.java
javac -d bin -sourcepath src src/ExpenseItem.java
javac -d bin -sourcepath src src/IncomeItem.java
javac -d bin -sourcepath src src/ReportGenerator.java
javac -d bin -sourcepath src src/SavingsGoal.java
javac -d bin -sourcepath src src/User.java
java -cp bin CLI
Error loading user data: financial_data.ser (No such file or directory)
Welcome to the Financial Tracker!

Money Buddy Main Menu:
========================
1. Manage Income
2. Manage Expenses
3. Set Savings Goal
4. View Financial Summary
5. Export Data
6. Exit
Enter your choice:
```

Now, how do you navigate through the Money Buddy Program itself?

1. Lets see how you can report your income to Money Buddy

- Choose option 1, enter your income source and income itself

```
Money Buddy Main Menu:
========================
1. Manage Income
2. Manage Expenses
3. Set Savings Goal
4. View Financial Summary
5. Export Data
6. Exit
Enter your choice: 1
Enter the income source:
IT Support
Enter the income amount:
350
Enter the income date (yyyy-mm-dd):
2023-01-18
Income added successfully.
```

2. Now, lets see how you can manage your expenses in Money Buddy

- Choose option 2, enter expense category, description and amount:

```
Money Buddy Main Menu:
========================
1. Manage Income
2. Manage Expenses
3. Set Savings Goal
4. View Financial Summary
5. Export Data
6. Exit
Enter your choice: 2
Enter the expense category:
Food
Enter the expense description:
Taco Bell
Enter the expense amount:
10
Enter the expense date (yyyy-mm-dd):
2023-01-20
Expense added successfully.
```

3. You can also set Savings Goal for each year and month in Money Buddy!

- Choose option 3, where you will have the option to set savings goal either as an amount or percentage for a specific year and month. For this use case analysis example, we'll choose to set savings goal as a percentage

```
Money Buddy Main Menu:
========================
1. Manage Income
2. Manage Expenses
3. Set Savings Goal
4. View Financial Summary
5. Export Data
6. Exit
Enter your choice: 3
Manage Savings Goal:
1. Set savings goal (amount)
2. Set savings goal (percentage of total income)
3. View current savings goal
4. Back to main menu
Enter your choice (1-4): 2
Enter the year and month for the savings goal (yyyy-mm):
2023-01
Enter the percentage of total income you want to set as your savings goal: 50
Savings goal set to 175.00 (50.0% of total income) for 2023-01
```

- After the Savings Goal is set, you can also view your savings goal for a specific year and month. We'll feed the command line two prompts in the following example.

3.1. Savings Goal for a month that wasn't set

```
Manage Savings Goal:
1. Set savings goal (amount)
2. Set savings goal (percentage of total income)
3. View current savings goal
4. Back to main menu
Enter your choice (1-4): 3
Enter the Year and Month for the savings goal (YYYY-MM):
2023-02
No savings goal set for 2023-02.
```

3.2. Savings Goal for a month that was set

```
Manage Savings Goal:
1. Set savings goal (amount)
2. Set savings goal (percentage of total income)
3. View current savings goal
4. Back to main menu
Enter your choice (1-4): 3
Enter the Year and Month for the savings goal (YYYY-MM):
2023-01
Savings goal for 2023-01:
Amount: 175.00
Percentage of total income: 50.00%
```

- Choose option 4 to back to the main menu

```
Enter your choice (1-4): 4

Money Buddy Main Menu:
==========================
1. Manage Income
2. Manage Expenses
3. Set Savings Goal
4. View Financial Summary
5. Export Data
6. Exit
Enter your choice:
```

4. After you have added all the required information you wanted to add as per your needs, you can check your financial summary for each specific year and month

- Enter option 4 in the main menu and then enter the year and month for the financial summary you would like to view
  For use case analysis, we'll look at two options the user can enter

4.1. No information added in financial summary

```
Money Buddy Main Menu:
==========================
1. Manage Income
2. Manage Expenses
3. Set Savings Goal
4. View Financial Summary
5. Export Data
6. Exit
Enter your choice: 4
Enter the year and month for the financial summary (yyyy-mm):
2023-09

Financial Summary for 2023-09:
Income: $.00
Expenses: $.00
Savings: $.00
No savings goal set for this month.
```

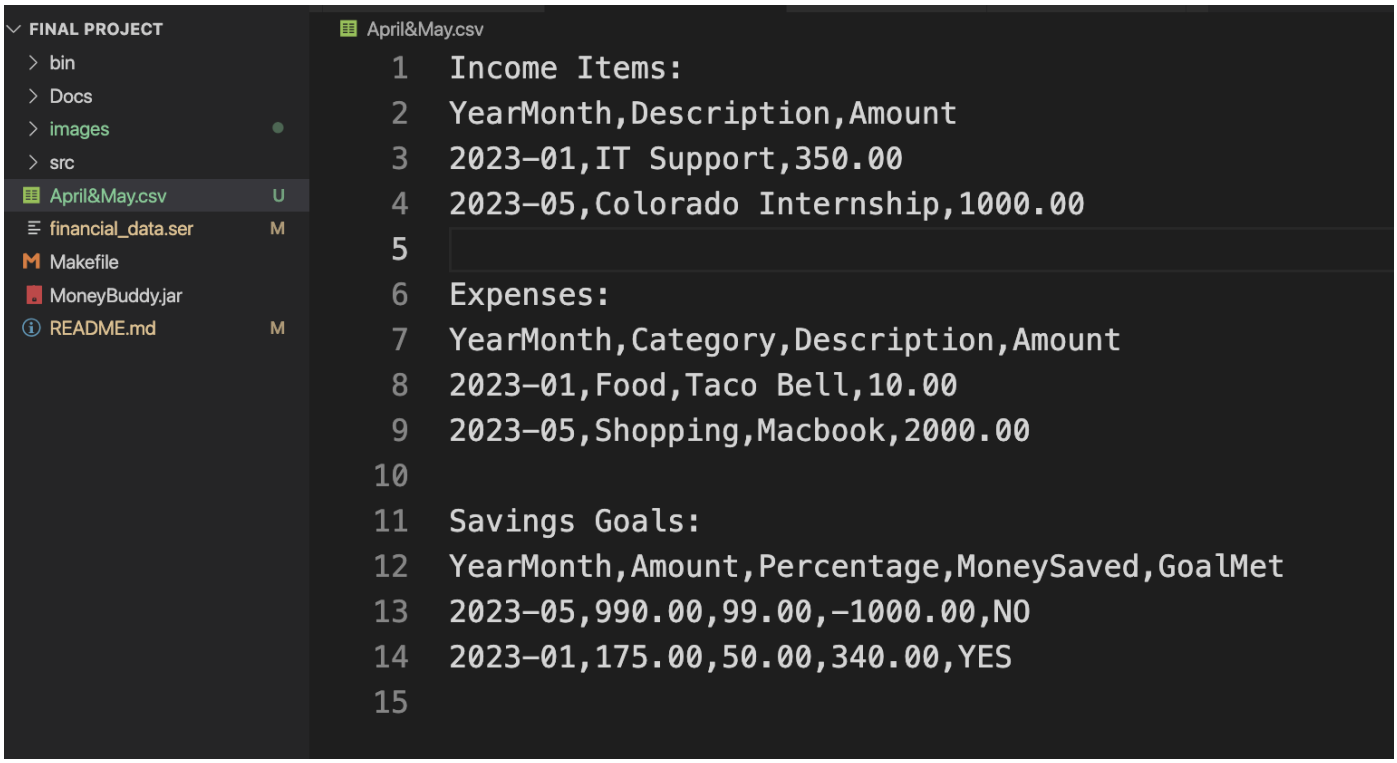
4.2. Information is in financial summary

```
Money Buddy Main Menu:
========================
1. Manage Income
2. Manage Expenses
3. Set Savings Goal
4. View Financial Summary
5. Export Data
6. Exit
Enter your choice: 4
Enter the year and month for the financial summary (yyyy-mm):
2023-01

Financial Summary for 2023-01:
Income: $350.00
Expenses: $10.00
Savings: $340.00
Savings Goal: 50.00% of total income ($175.00)

Congratulations! You met your savings goal this month.
```

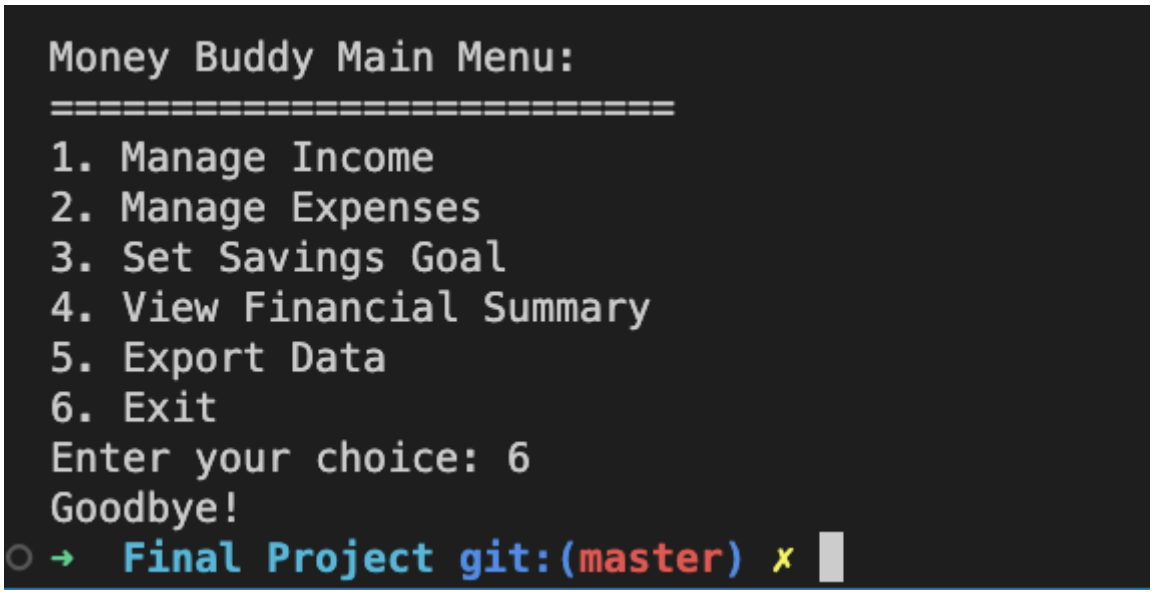
5. Lastly, you can export all this information in a `.csv` format to analyze all your financial reports. Enter 5 in main menu and enter name of the file `(e.g. April&May.csv)`

```
Money Buddy Main Menu:
========================
1. Manage Income
2. Manage Expenses
3. Set Savings Goal
4. View Financial Summary
5. Export Data
6. Exit
Enter your choice: 5
Enter the file name for the exported CSV file (e.g. 'report.csv'):
April&May.csv
Data has been exported successfully.
```

- You'll see the file in the project directory



6. If you are done using Money Buddy, enter 6 in the main menu. Don't worry, all your data will be saved!



Data Design

Money Buddy deals with the following data:

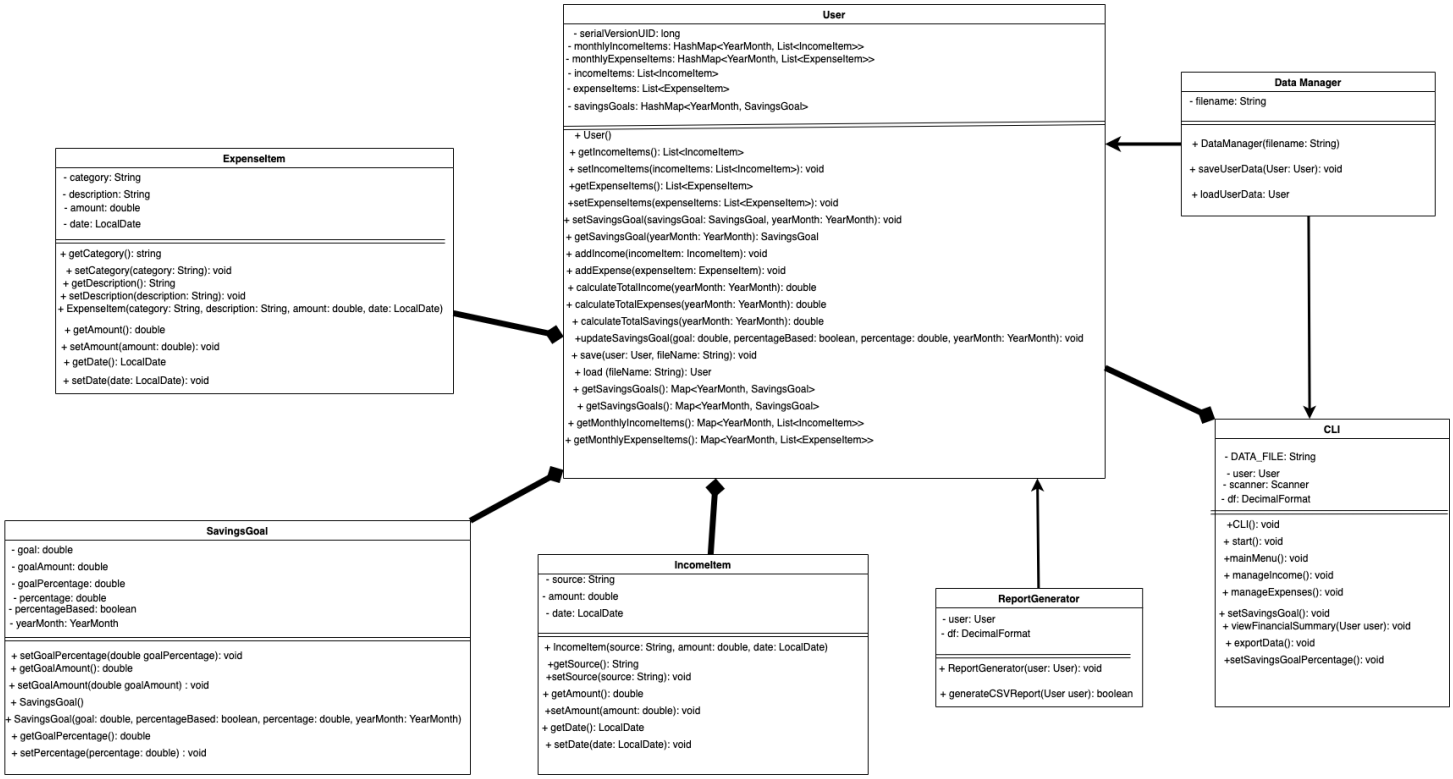
- User data *(User.java)* : The user's financial data, including income and expenses.
- Income data *(IncomeItem.java)* : The income source, amount, and date.
- Expense data *(ExpenseItem.java)* : The expense category, description, amount, and date.

- Savings Goal data *(SavingsGoal.java)* : The goal amount, and if it is percentage-based, the percentage of income, and the year and month.

Money Buddy also uses serialization to save and load the user data in a file `(financial_data.ser)`. This makes sure that the data is persistent

The User class manages the income, expenses, and savings goals. It provides methods for adding income and expenses, calculating totals, and managing savings goals. This shows that the data is aggregated into a larger structure within the `User` class.

Below is the UML Diagram for Money Buddy, which shows how data is represented and the basic data structure of how data may be related:



UI Design

Money Buddy has no UI for the time being and utilizes the Command Line Interface

Algorithm

Below is the basic Algorithm:

1. Initialize the program and load user data (if any)

2. Present the main menu with the following options to the user:

- Add income
- Add expense
- Set savings goal
- View financial summary
- Export data
- Exit program

To create a more visually pleasing CLI, use text formatting, ASCII Art, and clear navigation prompts.

3. Processing User Input

If the user selects 'Add income':

- Prompt the user to enter income details (source, amount, date)
- Validate and store the income data.
- Update the financial summary

If the user selects 'Add expense':

- Prompt the user to enter expense details (category, amount, date)
- Validate and store the expense data
- Update the financial summary

If the user selects 'Set savings goal':

- Prompt the user to enter a monthly savings goal option: Fixed amount or percentage of income
  - If the user chooses a fixed amount, the user enters a monthly savings goal in dollars
  - If the user chooses a percentage of income, the user enters a percentage which is validated. The savings goal in dollars is then calculated based on the entered percentage and total income
- Store the savings goal
- Update the financial summary

If the user selects 'View financial summary':

1. Retrieve the user's financial data:

- Access the stored data related to the user's income, expenses, and savings goal.

2. Calculate the financial summary:

- Total income: Sum all income sources to obtain the total income.
- Total expenses: Sum all expenses (by category or overall) to obtain the total expenses.
- Savings progress: Calculate the savings progress by subtracting the total expenses from the total income. Then compare the savings progress with the user's savings goal to determine if the goal has been met, exceeded, or not reached.

3. Format the financial summary and display it

If the user selects 'Export Data':

1. Prompt the user to select a file format:

- Display a list of supported file formats (e.g., CSV, PDF, etc.) and ask the user to choose one.

2. Display an error message if user inputs wrong choice
3. Generate the financial report:

- Retrieve the user's financial data (income, expenses, savings goal, etc.) and organize it into a structured format that can be easily converted to the chosen file format.
- Depending on the selected format, apply the appropriate formatting rules and convert the structured data into the desired file format. For example creating a CSV file with comma-separated values or a PDF document with formatted text and tables, or any other format-specific conversion.

4. Export the data

- Prompt the user to specify a file name and location to save the exported file, and save the file in that location
- Display a confirmation message if data exported successfully

4. Loop back to step 2 until the user chooses to exit the program

Below is the in-depth basic Algorithm for each class:

User.java class:

1. Data Members:

- `monthlyIncomeItems:` a HashMap that stores `YearMonth` as the key and a list of `IncomeItem` objects as the value.
- `monthlyExpenseItems:` a HashMap that stores `YearMonth` as the key and a list of `ExpenseItem` objects as the value.
- `incomeItems:` a List of IncomeItem objects.

- `expenseItems:` a List of ExpenseItem objects.
- `savingsGoals:` a HashMap that stores YearMonth as the key and a SavingsGoal object as the value.

2. Initializer (constructor):

- Initialize all data members with empty data structures (e.g., empty ArrayLists or HashMaps).

3. Define access methods for all data members (getters and setters):

- `getIncomeItems` and `setIncomeItems` for `incomeItems`
- `getExpenseItems` and `setExpenseItems` for `expenseItems`
- `getSavingsGoal` and `setSavingsGoal` for `savingsGoals`
- `getMonthlyIncomeItems` for `monthlyIncomeItems`
- `getMonthlyExpenseItems` for `monthlyExpenseItems`

4. All other additional methods:

- `addIncome` method for adding an IncomeItem object to the `monthlyIncomeItems` HashMap and updating the savings goal if necessary
- `addExpense` method for adding an ExpenseItem object to the `monthlyExpenseItems` HashMap
- `calculateTotalIncome` method for calculating the total income for a specific year and month
- `calculateTotalExpenses` method for calculating the total expenses for a specific year and month
- `calculateTotalSavings` method for calculating the total savings for a specific year and month
- `updateSavingsGoal` method for updating the savings goal based on the provided parameters
- `updateSavingsGoalBasedOnPercentage` method for updating the savings goal based on the original percentage for a specific year and month
- `save` method for saving a User object to a file
- `load` method for loading a User object from a file

IncomeItem.java class:

1. Data Members:

- `source` : a String representing the source of the income.
- `amount` : a double representing the amount of income.
- `date` : a LocalDate representing the date of the income.

2. Initializer (constructor)

- Accept three parameters: `source, amount, and date.`

- Assign the values of the parameters to the data members.

3. Define access methods for all data members (getters and setters):

- `getSource` and `setSource` for `source`
- `getAmount` and `setAmount` for `amount`
- `getDate` and `setDate` for `date`

ExpenseItem.java class:

1. Data Members:

- `category` : a String representing the category of the expense.
- `description` : a String representing the description of the expense.
- `amount` : a double representing the amount of the expense.
- `date` : a LocalDate representing the date of the expense.

2. Initiializer (Constructor):

- Accept four parameters: `category, description, amount, and date.`
- Assign the values of the parameters to the corresponding data members.

3. Define access methods for all data members (getters and setters):

- `getCategory` and `setCategory` for `category`
- `getDescription` and `setDescription` for `description`
- `getAmount` and `setAmount` for `amount`
- `getDate` and `setDate` for `date`

SavingsGoal.java class:

1. Data Members:

- `goal` : a double representing the savings goal amount.
- `goalAmount` : a double representing the current goal amount.
- `percentage:` a double representing the percentage of the savings goal.
- `percentageBased:` a boolean indicating whether the goal is percentage-based.
- `goalPercentage:` a double representing the goal percentage.
- `originalPercentage:` a double representing the original percentage value.
- `yearMonth:` a YearMonth object representing the year and month of the savings goal.

2. Initializers (Constructors):

- Default constructor which sets `goal` to 0, `percentageBased` to false, `percentage` to 0

- Constructor with `goal` parameter, sets `goal` to given value, `percentage` to 0 and `percentageBased` to false.

Constructor with `goal`, `percentageBased`, `percentage`, and `yearMonth` parameters:

- Set `goal` to the given value.
- Set `percentageBased` to the given value.
- Set `percentage` and `originalPercentage` to the given value.
- Set `yearMonth` to the given value.
- Set `goalAmount` to the given value.

3. Define access methods for all data members (getters and setters):

- `getGoal`, `setGoal` for `goal`
- `getGoalAmount`, `setGoalAmount` for `goalAmount`
- `getPercentage`, `setPercentage` for `percentage`
- `isPercentageBased` for `percentageBased`
- `getGoalPercentage`, `setGoalPercentage` for `goalPercentage`
- `getOriginalPercentage`, `setOriginalPercentage` for `originalPercentage`
- `getYearMonth` for `yearMonth`

DataManager.java class:

1. Data Members

- `filename` : a String representing the name of the file where user data is saved

2. Initializers (Constructors)

- Accept a `filename` parameter and set the filename data member to the given value.

3. `saveUserData` method:

- Accept a `User` object as a parameter.
- Create a `FileOutputStream` object with the `filename` data member.
- Create an `ObjectOutputStream` object using the `FileOutputStream`.
- Write the `User` object to the `ObjectOutputStream`.
- Close the `ObjectOutputStream`.
- Catch and handle any `IOException` that may occur.

4. `loadUserData` method:

- initialize a `User` object, user, to `null`.

- Create a `FileInputStream` object with the `filename` data member.
- Create an `ObjectInputStream` object using the `FileInputStream.`
- Read a `User` object from the `ObjectInputStream` and assign it to user.
- Close the `ObjectInputStream.`
- Catch and handle any `IOException` or `ClassNotFoundException` that may occur.
- Return the `User object.`

CLI.java class:

1. Data Members

- `DATA_FILE` : a constant String representing the file name for saving user data.
- `user` : a User object to manage income, expenses, and savings goals.
- `scanner` : a Scanner object for reading input from the command line.
- `df:` a DecimalFormat object for formatting currency values.

2. Initializer (Constructor)

- Initialize the User, Scanner, and DecimalFormat objects.
- Load the user data from the `DATA_FILE.`

3. `start` method:

- prints a welcome message and calls the `mainMenu` method

4. `mainMenu` method:

- Display main menu in a loop, read the user's choice, and based on the user choice call the appropriate method.

5. `main` method creates a new `CLI` object and calls the `start` method on the created object
6. `manageIncome` adds income items to the user
7. `manageExpenses` adds expense items to the user
8. `setSavingsGoal` manages the user's savings goals (amount or percentage based)
9. `viewFinancialSummary` displays the user's financial summary for a given year and month
10. `exportData` exports the user's data to a CSV file.
11. `setSavingsGoalAmount` sets an amount based savings goal (not percentage based)
12. `viewSavingsGoalAmount` displays the user's savings goal for a specific year and month.

ReportGenerator.java class:

1. Data Members:

- `User` object named `user` to manage income, expenses and savings goals.
- A `DecimalFormat` object named `df` for formatting currency values

2. `generateCSVReport` method takes a file name as a parameter and returns a boolean value. It also create a new `File, FileWriter, and PrintWriter` objects. Furthermore, it writes the headers and data for the Income Items, Expenses, Savings Goals sections. Lastly, it returns true if the report was generated successfully, and false otherwise. Also closes the `PrintWriter` and `FileWriter` objects

Future Improvements

In the future, additional features and enhancements could be considered:

- Implement Mult-user support and also enhancing data security measures to protect user financial information, such as implementing encryption, password protection, and more measures.
- Graphical User Interface (GUI) to improve user experience
- Using a database to store and manage user data more efficiently.
- Add customizable budget categories and subcategories to help users better organize their expenses and monitor their spending habits.
- Implement a feature to send users alerts or notifications when they are nearing their budget limits or when they have achieved their savings goals.
- Implementing real time multiple currencies and currency conversion by using some sort of API
- Introduce the ability to manage recurring income and expenses, such as monthly bills or salaries, to automate the tracking process and reduce manual data entry.
- Incorporate data visualization features, such as graphs and charts, to help users better understand their financial trends and make informed decisions.