Assignment 1 - The F.A.M. (Family Appointed Moderator)

Rahul Kukreja, Jeffrey Yim
jyim3@bcit.ca

BCIT CST - Computer Systems Technology Diploma

Introduction

You may work in groups of two (2) for this assignment. Create a team or join an existing team on github classroom. Make sure to include both your names and student numbers in comments at the top of driver.py.

The first COMP 3522 assignment for the semester is here!

This assignment will provide a platform for you to use the Python Fundamentals you have learnt so far while you stretch and flex your OOP muscles. You will get a chance to see how Object-oriented design can have a crucial role to play and an opportunity to implement the SOLID design principles.

Now, imagine if you will (or don't if this is already a reality for you) that you have a teenage child that is spoilt. This child tends to go on spending sprees and does not use their allowance and the salary from their part-time work responsibly. As a concerned parent you are worried that 2 years from now when your child leaves home to go to university, they won't have any savings to draw upon.

Enter the **F.A.M.**

The **Family Appointed Moderator (or F.A.M.)** is a parental control lock on an individual's bank account. In this assignment you will create a MVP (Minimum Viable Prototype) of the F.A.M. system. Your system will register a user, keep track of all the spending/transactions on their bank account and lock the account if certain conditions are met.

When going through the assignment brief start drawing out a preliminary UML class diagram to identify all the classes, attributes and any behaviors that you may need. This should be done before writing any code.

This assignment is quite open ended and designed that way so you can personalize the end result and have fun with it. Come up with interesting concepts and ideas. Be creative and enjoy the process! I encourage you to come discuss your designs with me if you want any feedback.

Submission Requirements

1. Clone the project from: **https://classroom.github.com/a/95DQwL32**
2. No late submissions allowed.
3. This is a group assignment. All code must be written by the group members only. I encourage you to discuss and share ideas with your friends but remember to write your own code!
4. Include a Readme file that describes how your application works and if there are any errors or use cases that have limitations or if it doesn't meet any of the requirements. This is also the spot where you want to document any features (the ones specified in this assignment and any extra features) that you may or may not have implemented. This will help me keep an eye out for them as I grade your work.

Grading

The assignment is marked out of **20**. For full marks, you must:

1. Correctly implement the requirements described in this document - **10 Marks**
2. Correctly format and comment your code. Eliminate all warnings offered by PyCharm, follow PEP 8 and PEP 257 guidelines, use good function and variable names, write code that is easy to understand, use whitespace wisely, write good and appropriate docstrings, etc. - **2 Marks**
3. Structure/design your classes to maximize code re-use, make it readable and maintainable. Follow the SOLID principles and the Law of Demeter. **- 4 Marks**
4. Submit a UML Class Diagram depicting appropriate use of OOP principles, inheritance and design in your code. **Auto-generated class diagrams will not be accepted** - 2 Marks
5. Handle errors and unexpected player behavior (read: input) gracefully **– 1 Mark**
6. Format your output. Make sure your messages display the correct information in a pleasant, readable manner. You could even use ASCII art if you dare! **– 1 Mark**

Implementation Requirements

The F.A.M. must implement the following features:

Registering A User

On startup, the user (usually a parent) must register their child's financial details. This includes (but is not necessarily limited to):

- The users name
- Age
- User Type (more on this below)
- Bank Account number
- Bank Name
- Bank Balance
- Their budgets (more on this below)

For testing purposes **you must** also include a load_test_users() method/function that loads a hardcoded test user. This will help you test your code and avoid entering user details every time you run the program.

You must still allow anyone using this program to register a user if they want to. The test method is for testing purposes only. Your program should also support multiple users and the functionality to switch between them.

Budget Categories

Each child that is being monitored is assigned the following budget categories. The exact value of each budget is assigned when registering the child as a user.

- Games and Entertainment
- Clothing and Accessories
- Eating Out
- Miscellaneous

A User Menu

Once the user account is set up and the budgets have been created, the system should prompt the user with the following menu options(or a variation of the following menu). The rest of the assignment brief explains how the app/each of these menu options should work.

1. **View Budgets**
   Selecting this option should show the user the current status of their budgets (locked or not) in addition to the amount spent, amount left, and the total amount allocated to the budget.

2. Record a Transaction

This should take the user to a sub-menu where they are prompted to enter the transaction details (refer to the "Record Transactions" heading below).

3. View Transactions by Budget

This should take the user to a sub-menu where they select their budget category and view all the transactions to date in that category.

4. View Bank Account Details

The application should print out the bank account details of the user and all transactions conducted to date alongside the closing balance.

Record Transactions

The application should maintain a collection of transactions which represent money going out of the users bank account. Provide the user an option to enter transaction details.

Each transaction should contain the following information:

- The timestamp the transaction was recorded (a nicely formatted datetime value).
- The dollar amount (positive, non-zero number).
- The budget category that this transaction belongs to.
    - Instead of prompting the user to enter the name of the budget category, provide them with a list of categories and ask them to select one.
- The name of the shop/website where the purchase took place.

The user should **not** be allowed to record a transaction if the transaction would cause their bank balance to go below zero. Additionally, the system should subtract the required amount from the users bank balance once a transaction has been recorded.

Depending on the type of user (more on this in the **User Types** section below), after a transaction has been recorded your system will want to perform checks to see if a warning or notification should be issued. A list of transactions that have taken place in a budget category should be printed out to the console if:

- The user receives a warning that they are getting close to exceeding their assigned budget for the category in question
- The user receives a notification that they have exceeded their assigned budget for the category in question.

The transactions printed should be the transactions pertaining to the budget category in question. That is, if the user gets a warning that they are about to exceed their budget for "Games and Entertainment", then any transactions belonging to this category should be printed for review.

Lock Out

The app has the ability to lock a user out of recording transactions (and effectively spending any money) based on certain conditions as specified by their User Type (more on this in the User Types section below).

If a user has been locked from a budget category:

- They should be notified of this via a console message.
- Any attempt at recording transactions in the affected budget category should be denied.

User Types

Every family and every child is unique. The F.A.M. prototype recognizes this face and supports different types of users. The app provides different moderation levels for each user type.

The Angel

The Angel represents a user whose parents are not worried at all. This child has never (as far as their parents are concerned) broken a single rule. They already have a five-year plan in place and a roadmap which is guaranteed to get them into Harvard. The Angel is the child who would set up their own FAM account so they can monitor their expenses.

This user type:

- Never gets locked out of a budget category. They can continue spending money even if they exceed the budget in question.
- Gets politely notified if they exceed a budget category.
- Gets a warning if they exceed more than 90% of a budget.

The Troublemaker

The Troublemaker represents a user who often finds themselves in... well.. trouble. These are usually minor incidents and their parents are concerned but not worried. Parents usually set up a FAM account to monitor their expenses and impose light restrictions.

This user type:

- Gets a warning if they exceed more than 75% of a budget category.
- Gets politely notified iIf they exceed a budget category.
- Gets locked out of conducting transactions in a budget category if they exceed it by 120% of the amount assigned to the budget in question.

The Rebel

The Rebel represents a user who refuses to follow any rules and believes that society should be broken down and restructured. They do not want to pursue "*a standard education*", "*conform to the economic/capitalist foundations of society*" or "*get a job*". Parents of these children are quite worried and turn to F.A.M. when they are out of options.

This user type is strictly monitored:

- They get a warning for every transaction after exceeding 50% of a budget.
- Gets ruthlessly notified if they exceed a budget category.
- Gets locked out of conducting transactions in a budget category if they exceed it by 100% of the amount assigned to the budget in question.
- If they exceed their budget in 2 or more categories then they get locked out of their account completely

Concluding Thoughts

Remember to approach your code in an object-oriented fashion. Don't just start bashing out code and hoping that it will all work out. Consider these steps.

1. Identify the classes and objects you will need. Figure out what each class is responsible for.
2. Draw a UML class diagram showing how the classes/objects relate to each other. Write down the attributes and methods.
    a. At this stage you can do a simple sketch on paper. Don't worry about syntax and only mention the important attributes/methods.
3. Write some code
4. Repeat from step 1. (Take an iterative approach!)

When creating your final UML diagram, be sure to check syntax, and mention all the attributes and methods.

- Ensure you push your work to github classroom. I'd like to see sensible git commits comments, and commits must take place at logical points in development.

That’s it. Good luck, and have fun!