**Project Requirements**

Problem:

Organizing expenses and incomes is necessary for responsible financial habits, yet is problematic unless it can be layed out in an organized fashion.

Code must accept user input of financial data of different forms. It must accept expenses and incomes. It must accept different types of incomes and expenses such that the code will be able to separate the data accordingly into categories. It must take into account and calculate income based on hourly wage data, etc. The system must prompt the user to choose to create a dynamic amount of incomes and expenses of various types and must organize the data accordingly into files. The system must be used within a C++ compiler, and the user must have basic competence in following directions and locating the output files.

Solution:

The way that this project solves the problem is by organizing incomes and expenses into clearly organized output files, so that a user can better organize and view financial data.

Design:

I will create two parent classes: Income and Expense.

User will be prompted to create Incomes and Expenses.

Upon construction of an Income, the constructor will prompt the user to initialize parameters, using terminal output prompts and terminal input.

Income will include fundamental properties of incomes, shown in UML

Income will have the child classes: Silverwood and PhotoAndVideo.

Silverwood will use different terminal prompt language upon construction than its Parent constructor and must, therefore, bypass the code in the parent constructor. (Of course, the parent constructor will be called automatically; I will use a bool parameter in the parent constructor which will allow either a complete execution or a complete bypass of its statements).

Silverwood will include tips, optionally.

PhotoAndVideo will also bypass the code in the parent constructor so that it may provide a completely different user interaction.

User determines whether PhotoAndVideo is video or photo income.

Photo and Video will, optionally, incur Expenses set by the user.

PhotoAndVideo will calculate profit accordingly.

By the above, I am usefully using polymorphism, as to incur specifically different construction processes for child classes.

Expense will have the child classes: Recreation and College

Recreation will construct similarly to Expense. Yet, it will exist as a separate type of object, and thus be easily separable from Expense objects for purposes of organizing types of expense.

College classes will possess more members than Expense objects and construct differently.

Child objects must output differently into files. Thus, I will either use operator overloading or polymorphism function override (probably with a toString() function) to create different types of outputs per type of class. Outputs will all be to corresponding txt files.

This is another useful feature of my usage of polymorphism in this project.

The system must create dynamic memory and utilize different types of streams. It will require that the user has done initial preparations for their inputs; the user must know income/expense metadata.

The user is assumed to be planning finances for the year.

The main function will include functions which will allow the user to load vectors with expenses and incomes. The vectors will be used to store and write data to respective files.

Upon future improvement of the code, the vectors can be accessed in order to do more advanced calculations and manipulations of the objects within.

Expense

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

#string name //Name

#float amount //Amount

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

+Expense(bool isDerived) //Prompts user and constructs object.

// bool isType is used to bypass statements

// if a specific type of Expense is invoked.

+string toString() //Generates string of data

+float getAmount() // Returns amount

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Recreation\_Expense

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

+Recreation()//Constructor

College\_Expenses

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

-float Tuition

-float Housing

-float Books

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

+College()//Constructor

string toString()// Overrides the Expense toString() function

Income

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

#bool hourly

#double wage

#double flatAmount

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

+Income()

+string toString()// Generate a string of data

PV

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

-bool isPhoto //Is photo?

-vector<Income> fees //Extra fees charged to clients

-vector<Expense> costs //Costs incurred

-float profit // Profit

-float totalCost // Costs incurred

-float totalFees // Total fees

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

+PV()

+toString()

Silverwood

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

-int weeksPerYear // Weeks per year worked

-int daysPerWeek // Days per week worked

-float tipAvg //Avg tips earned per day

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

+Silverwood() //Constructor