Programing2.5 - Course Project

Ali Shahrestani

▼ Deliverable 1 Instructions

Submit a project idea, addressing the following points:

- Scenario: Explain the scenario under which your project will operate.
- Design Paradigm: List the functionalities you plan to demonstrate.
- Expected Output: Describe the expected results and the actions the user can perform with your application.
- Specify the hierarchies (at least two) of the project
- Specify the what the interface is and why do you need it in your project
- Specify which methods apply runtime-polymorphism
- Specify in which class and for what purpose textIO will be used.
- Specify in which class(es) `Comparable` will be implemented, and what class(es) need `Comparator`
- A class diagram to show the classes and interfaces of the project and their relationship.
- Specify which part (class, interface and methods) will be implemented for deliverable 2 (50%)
- Git Repository: Initialize a Maven project with valid `.gitignore`, and a `README.md` file for a project description. Create a `doc` folder which contains Deliverable 1 PDF.

Deliverable 1

Scenario

The project is called “Organization Budget Tracker”.

It is a fun poke at the financial operations that needed to be done when organizing VanierHacks 2025. This project is a much simplified version of different software used to track expenses and reimbursement.

The scenario/purpose of this project is for any organization (eg. a club or an association) that needs to buy things and reimburse those who actually made the purchases, all whilst keeping track of how much of the budget remains.

The program will be used by everyone in the organization but certain actions are reserved only for certain positions. For example, anyone can submit a request for reimbursement but only the treasurer can approve the transfer of funds.

The user must login to use the program.

Design Paradigm

The current planned functionalities (subject to change) are:

System Administrator

- Manage user accounts
    - View all users
    - Create a new user
    - Delete a user
    - Change a user’s type
- Configure sources
    - Create accounts
    - Create projects
    - Create budget
- View technical details (extra)
- View system audit logs (extra)

Treasurer

- Manage sources
  - View all accounts and their details
  - View all projects (events that need funding)
  - View all budgets (income sources) and their details
  - Update the amount of a budget (extra)
  - Edit projects details (extra)
- View financial audit logs (extra)
- Manage reimbursement
  - View all reimbursement requests
  - Approve reimbursement requests
  - Edit reimbursement requests (extra)

Organizer

- View their account
  - View the history of all approved reimbursements deposited into their account
- Request reimbursement
  - View the status of the pending reimbursement
  - View history of reimbursement
  - Modify pending reimbursement requests (extra)
- View all active events

Some functionalities overlap who can do them.

The functionalities marked as “extra” are low priority ones and the decision to implement them or not will be made when the other functionalities have been made.

Expected Output

All users will interact with the program with through the console. There is no graphical user interface.

User input will be heavily facilitated by the separate Java Scanner Menu Framework (provides input/error handling and prompt reuse).

The program starts with a default sys admin account. The sys admin must then create accounts for all other users.

Users must first login before being able to perform actions outlined in the previous section.

Users will be able to submit reimbursement request and track their status.

The treasurer will be able to view pending requests, approve/reject them, and monitor the budget.

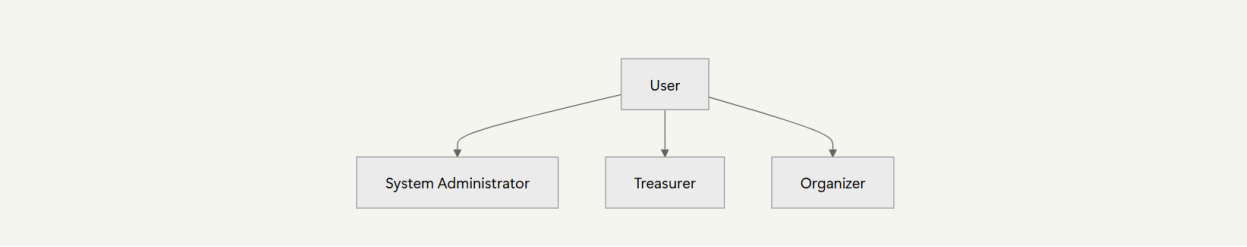
The program's data will be saved in different folders and files. The data will be loaded when the program starts or when needed by their related actions.

The program determines which organization to load based on the first command line argument passed which specifies the location of the organization's files.

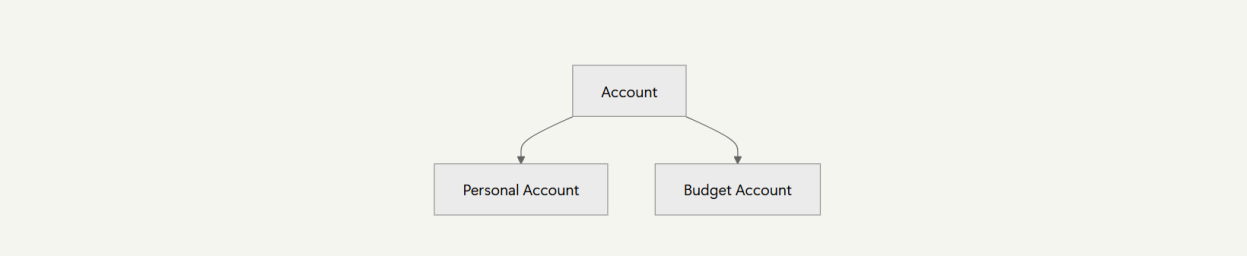
Concepts

Hierarchies

User Hierarchy



Account Hierarchy



There will be more classes (eg `Project`, `Transaction`, `Expense`, `Reimbursement`, etc).

Interfaces

Transactable

This will be an interface with `

The `Savable` interface will declare a `export()` and a `import()` method to be used by the `DataManager` or its subclasses. This will be implemented by almost all the object classes to self-implement the necessary steps to save/load the object.

There may be more interfaces implemented later on.

Polymorphism

Polymorphism will be demonstrated in the `displayUserInfo()` method. This method will be defined in the `User` class and overridden in the subclasses to display user-specific information (eg regular organizers have their personal accounts, there will be info on that).

There will be more instances of polymorphism.

TextIO

TextIO will be used in the `DataManager` class (and the subclasses pertaining to different parts of the system).

The class will handle reading, loading, and writing user data, reimbursement records, and budget information for data persistence.

The class is also crucial for initial program startup to load information about the organization and its users.

Some compartments of the program will save data in a single file.

Some compartments of the program will save data per line in a file.

Some compartments of the program will use a whole directory and separate data into individual files.

Comparison

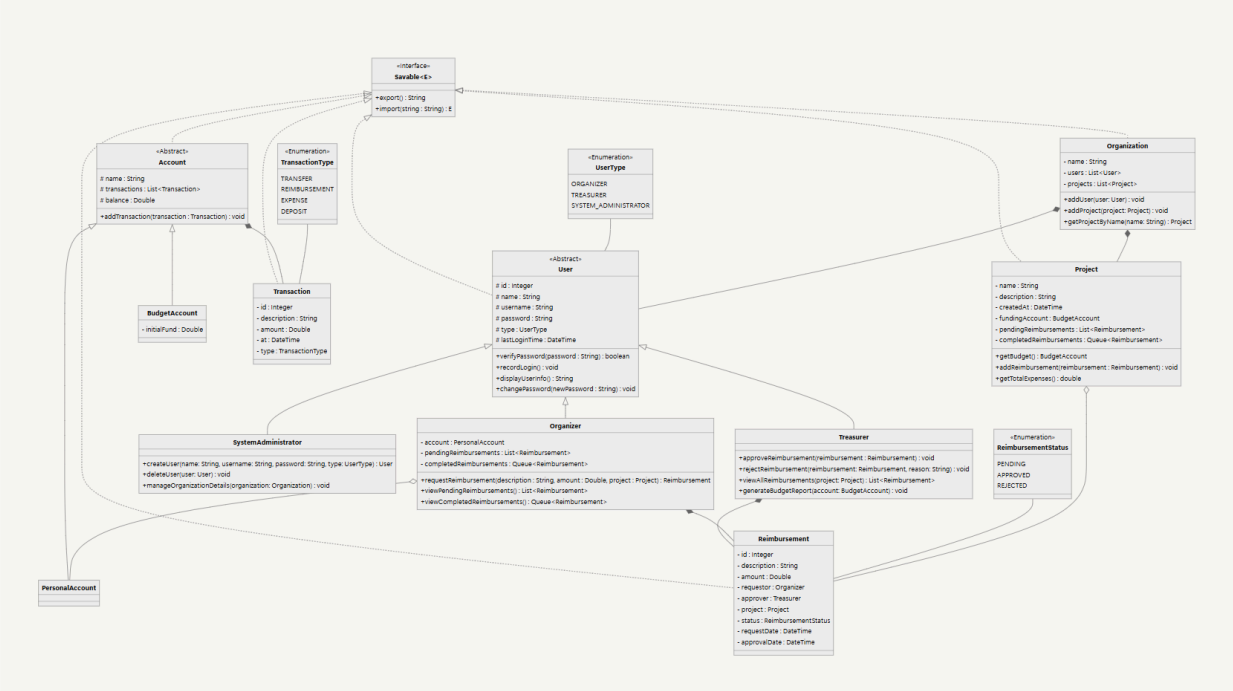
The `User` class will implement the `Comparable` interface to be able to sort users based on their permission level in the organization.

The `Project` class will implement the `Comparator` interface to be able to sort projects based on different criteria such as:

- Amount of money used on the project
- Number of reimbursement for the project
- Number of unique people reimbursed for the project

There may be more comparisons implemented.

Class Diagram



This diagram can be viewed as an image in mermaid-diagram-2025-04-19-212150.png

Note the following implications of the UML diagram above:

- Auto generated methods such as constructors, getter/setters, equals, etc are not included
- Abstract methods are only included in the abstract class
- Abstract interface methods are only included in the interface
- Quantifiers (cardinality / multiplicity) are not defined
- The diagram makes no guarantees on the order of fields and methods
- Utility classes are not showed
- User intractability (”flowchart”) classes are not showed
- Additional annotations follow https://mermaid.js.org/syntax/classDiagram.html

A full class diagram will be available for subsequent deliverables. The diagram above is subject to change and is only an initial plan.

Priority for the 50% milestone is to implement `User` and its subclasses, `Account` and its subclasses, `Transaction` and `Project`.

To further clarify, creating the conceptual classes and their methods (business logic) will be prioritized over “flowchart” classes and their methods (user intractability with the program). Data preservation will be the third area of concern.

And all Enums related the classes mentioned.

GitHub

The project will be stored on GitHub at https://github.com/VanierCollege/Programing2.5-CourseProject/.

Diagrams