Design Pattern: Expense Tracker Software

Contents

[Document Information 1](#_Toc183957460)

[List of Authors 1](#_Toc183957461)

[Revision History 1](#_Toc183957462)

[Chosen Design Patterns 1](#_Toc183957463)

[Singleton 1](#_Toc183957464)

[Observer 2](#_Toc183957465)

[Strategy 3](#_Toc183957466)

[Design Pattern on UML <Check Design\_Pattern.drawio> 3](#_Toc183957467)

# Document Information

## List of Authors

|  |
| --- |
| Names |
| Charles Eboson |

## Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Date | Author | Notes |
| 0.1 | 01.12.2024 | Charles Eboson | First draft. |

# Chosen Design Patterns

## Singleton

The **Database** class ensures that only a single instance of the class is created and shared across the entire application, avoiding multiple connections to the same database and ensuring consistency.

Classes Involved:

* **Database: Central point for database connection**
* **TransactionManager**: For managing transactions (add, modify, delete, etc.).
* **TransactionList (Abstract)**: For retrieving and managing transaction records.
* **DataStorage**: Handles saving and loading of user progress, settings, and transaction data.
* **Settings**: For managing application configurations (e.g., preferred currency).
* **UserAccountManager**: Handles storing, retrieving, and updating user account details in the database for the AuthManager class to authenticate and authorize users.

This ensures that only one database connection is shared across the system.

## Observer

The Observer pattern ensures that when there are changes in one object (the subject) all its dependents (observers) are equally notified and updated automatically.

Classes Involved:

**TransactionManager** (Subject):

* The **TransactionManager** manages the list of transactions and notifies observers when transactions are added, removed, or updated.
* It holds a list of **Observer** objects and calls their update() method whenever there is a change in the transactions.

**Observer** (Interface):

* The **Observer** interface defines the update() method, which is called whenever the **TransactionManager** notifies its observers about a change. Each observer (like **TransactionHistory**, **ExpenseSummary**, or **ReportSummary**) will implement this method and define how they update based on the changes.

**TransactionHistory, ExpenseSummary, ReportSummary** (Observers):

These are concrete observers that implement the update() method from the **Observer** interface. They respond to changes in the transactions:

* **TransactionHistory**: Updates and reloads the transaction history.
* **ExpenseSummary**: Recalculates the expense summary.
* **ReportSummary**: Regenerates the report summary.

This ensures that whenever a transaction is modified, the observers automatically update themselves without the need for explicit calls from TransactionManager.

## Strategy

The Strategy Pattern defines a family of algorithms, encapsulates them, and makes them interchangeable. In the context of the expense tracker app, this pattern can be applied to switch seamlessly between different filtering options for viewing recent transactions. Additionally, it can be utilized for generating report summaries in various formats, such as monthly or yearly reports.

Classes Involved:

**FilterStrategy** (Interface for Recent Transaction Filtering):

* Defines the filter() method for filtering transactions.

**ReportStrategy** (Interface for Report Generation):

* Defines the generateReport() method for generating reports.

This is important because new filtering or report generation algorithms can be easily implemented without modifying the existing code.

# Design Pattern on UML <Check Design\_Pattern.drawio>