PFS Refactoring Report

Refactoring 1: THE DATA ACCESS LAYER

THE PROBLEM | previously the data access layer was one class which has similar functions for each expenses, labels, and pay to. There was a separate function for getting the ids, creating a new entry, retrieving an entry via id, and updating an entry via id. The reduplication of code make it very tedious and problematic when making changes to the database layout or adding new functionality.

THE SOLUTION | the solution was to change how we thought and worked with the database. Previously we only thought of the database as one big collection of various data. We changed to look at the database as a collection of tables, the data itself did not matter. This allowed us to create the Database Table abstract class which gave us all the functionality we needed to interact with any part of the database. For each table in the database, all we had to do was override the Database Table class, define the convert to object, add, update, and delete methods.

Refactoring 2: System Manager Classes

THE PROBLEM | previously there were three separate manager classes used in the system: Expense Manager, Label Manager, and Pay-To Manager. All three classes had identical functionality but only varied in which part of the database they accessed. This once again made it very hard to change anything since any change in behavior had to be replicated in each manager class.

THE SOLUTION | after the refactoring to the data access layer, this was very easy to change by employing the strategy design pattern. Now that each part of the database was accessible through the same methods via a common parent class, all three manger classes could be replaced with just one. All three manager classes were removed and replaced with the Manager class. This class takes a Database Table and provides all the necessary methods to interact with the table. The PF System (the main system class) now initializes three instances of the Manager class, one for each table, instead of having to initialize three different classes to do the same job on different data.

Refactoring 3: Cache Layer

THE PROBLEM | with the high number of database calls the whole system was slowing down, specifically in the data mining features.

THE SOLUTION | we introduced an additional layer in our overall design. Between the Data Access Layer and the Database Layer we created the Cache Layer. All communication with the database goes through the cache. If the object is available in the cache we don't access the database, thus saving us the cost of IO and processing used in a database query. When writing to the database, we also write to the cache so that we decrease the chance of accessing the database soon.