Framework - Observable Party Account

## Team: Blue [Md Avishek Ahmed, Md Sajedul Islam & Achal Jain]

**Frameworks operations and solutions by Patterns**

|  |  |  |  |
| --- | --- | --- | --- |
| **Operations** | **Common** | **Variation** | **Pattern Used** |
| **Customer Creation** | Personal, Company | Personal, Company | Party |
| **Account Transaction** | Entry | Debit & Credit | Account |
| **Account behavior & Interest Selection** | Add Interest | Save-in, Check-in, Gold, Silver, Bronze | Strategy |
| **Finance Handler** |  |  | Facade |
| **Notification Rule** | Notification | Different Rules | Functor |
| **GUI Generation** | AGui | Gui elements | Template Methods |
| **Type Generation** |  | Customer types generation | Factory Methods |
| **Reporting** | Report Generation | Different Report Types | Functor |

# Why & Where Pattern are Used:

**Party Pattern**:

**Reason**: In our framework there is person and Company. They have many common and some different behaviors. Therefore, we put any behavior that is common to Personal and Company units on Party. It helps us to achieve the DRY and manage simplify the operations.

**Classes**: ICustomer, IPerson, ICompany, Customer, Company & Person

**Account Pattern:**

**Reason**: As our framework need to create different account and every account is always updating with transactions and those transaction histories are required that is why we have used Account Pattern. We implement DebitEntry and CreditEntry to serve all financial transactions because financial transactions are either debit or credit.

**Classes**: Account, Entry [DebitEntry & CreditEntry]

**Functor Pattern**:

**Reason**: While developing the framework we use functor Pattern for different notification rules and different reporting types.

**Classes**: IReport, TxtReport, NotificationRule, CompanyNotificationRule, PersonalNotificationRule,

**Strategy Pattern**:

Reason: In our Framework we have select the account behaviors like Save-in or Check-in in runtime and also we select the interest dynamically. In these sections we use strategy pattern

**Classes**: Type, SavingType, CheckinType.

**Template Pattern**:

**Reason**: Our framework has AGui and it has three parts Creation, Operation and View. Those parts are generated by template method loadView() and sub-operation of loadView() is override by sub gui classes.

**Classes**: AGui, DefaultGui, BankGui.

**Facade Pattern**:

**Reason**: we use Façade to hide all the complexity from the outer world. It provides us a single interface for all functionality.

**Classes**: FinanceHandler.

**Factory Methods:**

**Reason**: As we do not know that type of Account Type will be used that is why we are hiding it by Factory Methods.

**Classes**: FinanceHandler.

# Tools:

**Source Code Management**: GitHub

**IDE**: Eclipse

**Design**: StarUML