Name: Saloni Ingle

SJSU ID:017506294

INDIVIDUAL PROJECT REPORT

1. Describe the problems you are trying to solve.

Problem Statement: Currency Transaction Processor

Objective:

To develop a Java application to process currency transactions. The application will read transaction details from an input file (JSON, CSV, or XML), validate the currency codes, convert the amounts to the target currency based on predefined exchange rates, handle errors gracefully, and generate an output file in the same format as the input.

Key Features:

Input File Parsing: Read transaction details from an input file.

Currency Code Validation: Validate currency codes against a predefined list.

Exchange Rate Retrieval: Retrieve exchange rates between different currencies.

Currency Conversion: Convert amounts from the original currency to the target currency.

Error Handling: Record and handle validation and conversion errors.

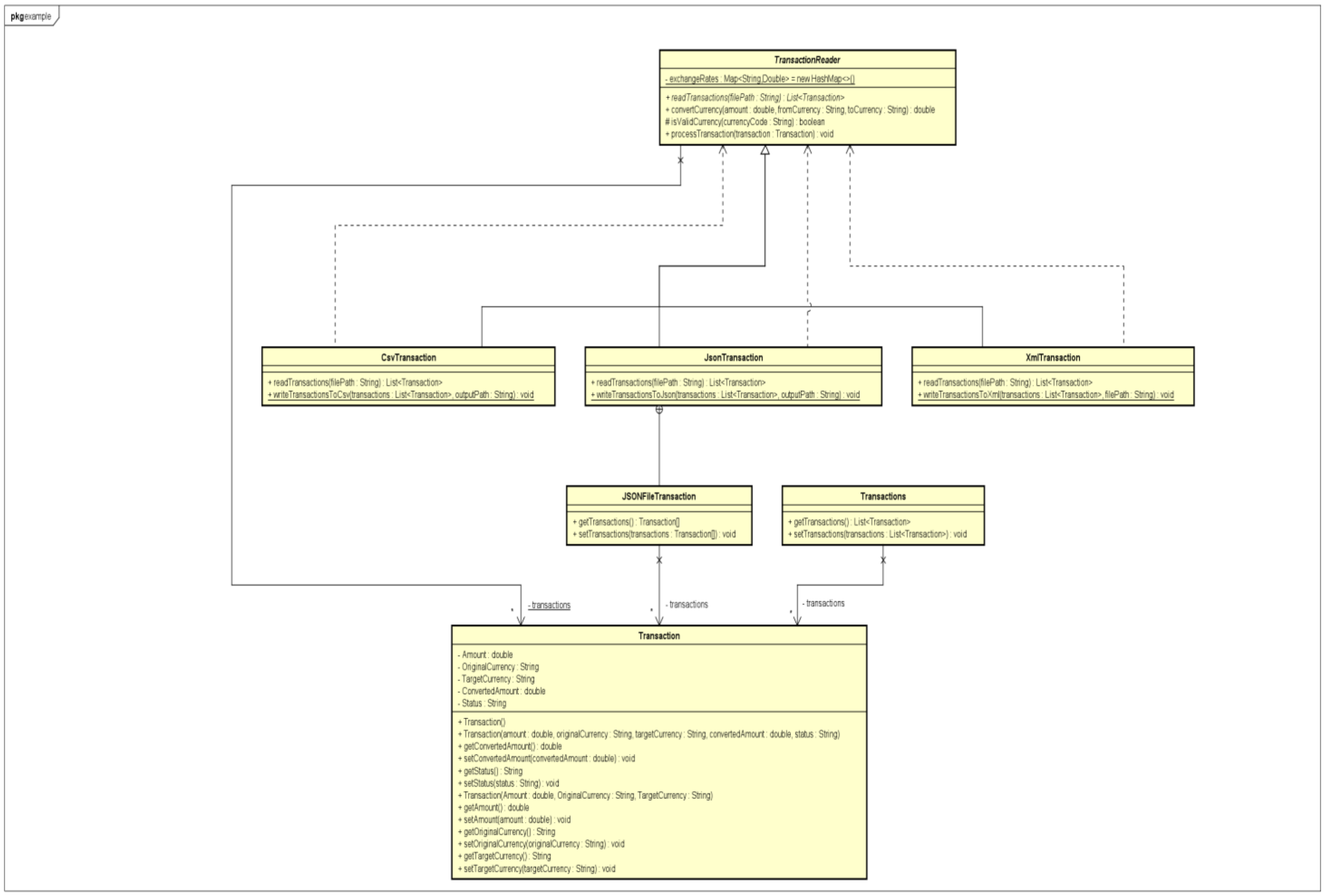
Output File Generation: Write processed transactions to an output file in the same format as the input.

Project Management: Use Maven as the build tool for efficient dependency management and project structure.

Testing: Using Maven JUNIT testing for automated testing of the project.

2. Describe what design pattern(s) you want to use (use plain text and class diagrams). Justify the choice of design pattern(s)

I am using Template Design Pattern is suitable for this project because it provides a framework for defining the structure of an algorithm while allowing subclasses to customize certain steps of the algorithm without changing its overall structure. This promotes code reuse, maintainability, and flexibility in adapting to changing requirements or evolving future needs. Here's how I applied Template design pattern to different aspects of the currency transaction processing project:


pkg example
TransactionReader
- exchangeRates : Map<String,Double> = new HashMap<>()
+ readTransactions(filePath : String) : List<Transaction>
+ convertCurrency(amount : double, fromCurrency : String, toCurrency : String) : double
# isValidCurrency(currencyCode : String) : boolean
+ processTransaction(transaction : Transaction) : void
CsvTransaction
+ readTransactions(filePath : String) : List<Transaction>
+ writeTransactionsToCsv(transactions : List<Transaction>, outputPath : String) : void
JsonTransaction
+ readTransactions(filePath : String) : List<Transaction>
+ writeTransactionsToJson(transactions : List<Transaction>, outputPath : String) : void
XmlTransaction
+ readTransactions(filePath : String) : List<Transaction>
+ writeTransactionsToXml(transactions : List<Transaction>, filePath : String) : void
JSONFileTransaction
+ getTransactions() : Transaction[]
+ setTransactions(transactions : Transaction[]) : void
Transactions
+ getTransactions() : List<Transaction>
+ setTransactions(transactions : List<Transaction>) : void
- transactions
- transactions
- transactions
Transaction
- Amount : double
- OriginalCurrency : String
- TargetCurrency : String
- ConvertedAmount : double
- Status : String
+ Transaction()
+ Transaction(amount : double, originalCurrency : String, targetCurrency : String, convertedAmount : double, status : String)
+ getConvertedAmount() : double
+ setConvertedAmount(convertedAmount : double) : void
+ getStatus() : String
+ setStatus(status : String) : void
+ Transaction(Amount : double, OriginalCurrency : String, TargetCurrency : String)
+ getAmount() : double
+ setAmount(amount : double) : void
+ getOriginalCurrency() : String
+ setOriginalCurrency(originalCurrency : String) : void
+ getTargetCurrency() : String
+ setTargetCurrency(targetCurrency : String) : void

Class Diagram

As shown in the above diagram;

a. TransactionReader is the abstract class for processing transactions. This class includes methods for parsing InputFile, validating CurrencyCodes, converting Currency and processing transaction.
b. XMLTransaction is the subclass of the TransactionReader class.This class includes methods for reading of the input file and writing of the output file in the same format as the input file(i.e “.xml”)

c. JSONTransaction is also the subclass of the TransactionReader class.This class includes methods for reading of the input file and writing of the output file in the same format as the input file(i.e ".json")
d. CSVTransaction is also the subclass of the TransactionReader class.This class includes methods for reading of the input file and writing of the output file in the same format as the input file(i.e ".csv")
e. Transaction class encapsulates the details of a transaction, including the amount, original and target currencies, conversion status, and converted amount. It provides methods for accessing and updating these attributes.
f. Transactions class acts as a wrapper for a collection of Transaction objects.

3. Describe the consequences of using these pattern(s).

a) Complexity and Indirection: Introducing an additional layer of abstraction through the template class may increase the complexity of the codebase. We need to understand the interaction between the template class and its subclasses, leading to potential indirection and cognitive overhead.

b) Inheritance Overuse: The Template Design Pattern relies on inheritance to define the structure of the algorithm. While inheritance can promote code reuse, it can also lead to issues such as tight coupling, brittleness, and difficulty in maintaining class hierarchies.

c) Limited Runtime Flexibility: Since the overall algorithm structure is defined in the template class, runtime flexibility may be limited. Changes to the processing logic or algorithm structure may require modifications to the template class or its subclasses, potentially impacting existing code.

d) Overhead of Abstraction: Introducing a template class and abstract methods adds a level of abstraction to the design. While abstraction can improve code maintainability and flexibility, it may also introduce additional overhead in terms of design complexity and performance.

e)Limited Functionality: This pattern limits the future scope functionality of this project .This pattern will not work if we want to generate output file format different from the input file format i.e converting from csv input file to json output file.

In summary, while the Template Design Pattern offers several benefits in terms of modular design, code reusability, and customization, it also comes with challenges related to complexity, inheritance, runtime flexibility, and abstraction overhead. Careful consideration and design are necessary to leverage the pattern effectively and mitigate its potential drawbacks.