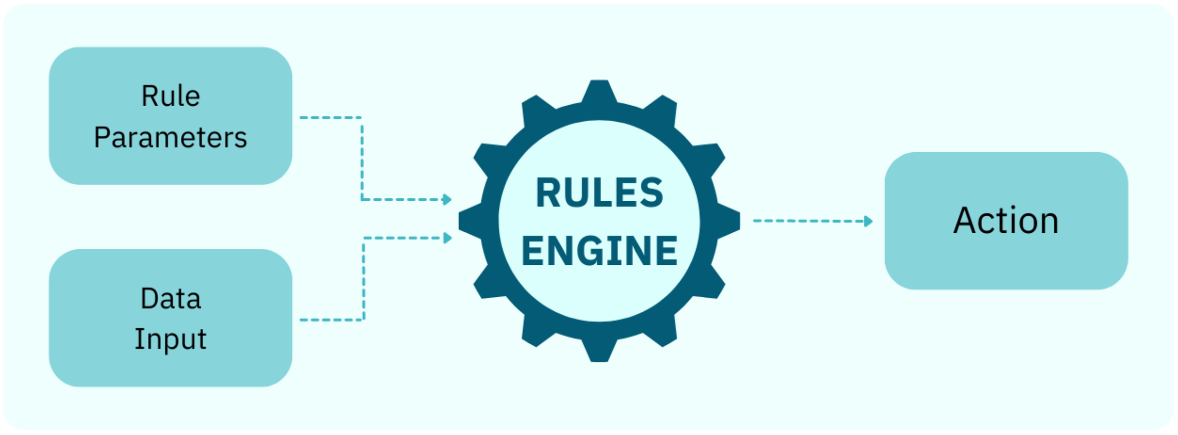
Rule Engine

Scala

PROJECT OVERVIEW

The project aims to develop a rule engine in Scala for a retail store. This rule engine qualifies orders' transactions for discounts based on a set of qualifying rules. Additionally, it automatically calculates the proper discount based on specific calculation rules.



RULES REQUIRED

The project implements the following qualifying rules and calculation rules:

QUALIFYING RULES

1. **Less Than 30 Days Remaining for Product Expiry:**
    - If the product has less than 30 days remaining to expire from the day of the transaction, it qualifies for a discount.
2. **Cheese and Wine Products On Sale:**
    - Cheese products qualify for a 10% discount.
    - Wine products qualify for a 5% discount.
3. **Special Discount on 23rd of March:**
    - Products bought on the 23rd of March qualify for a 50% discount.
4. **Quantity-Based Discount:**
    - If a customer buys more than 5 of the same product:
        - 6-9 units qualify for a 5% discount.
        - 10-14 units qualify for a 7% discount.
        - More than 15 units qualify for a 10% discount.
5. **Sales Through App:**
    - Sales made through the app qualify for a special discount based on the quantity rounded up to the nearest multiple of 5.

6. **Sales Using Visa Cards:**
    - Sales made using Visa cards qualify for a 5% discount.

CALCULATION RULES

- Transactions that didn't qualify for any discount will have a 0% discount.
- Transactions that qualify for more than one discount will get the top 2 and get their average.

APPROACH FOLLOWED IN THE CODE

Core Functional Logic:

1. toOrder(line: String): Order

- **Functionality:** Converts each line of the input CSV file to an **Order** object.
- **Description:** Parses the CSV line, extracts relevant information, and creates an **Order** object.

2. Qualifying Rule Functions:

- **Functionality:** Determine if an order qualifies for a specific discount based on certain conditions.

1. **isExpiringSoon(order: Order): Boolean**
    - **Description:** Checks if the remaining days for the product's expiry is less than 30 days.
2. **cheeseAndWineOnSaleQualifier(order: Order): Boolean**
    - **Description:** Checks if the product is either cheese or wine.
3. **specialDiscountQualifier(order: Order): Boolean**
    - **Description:** Checks if the order date is on the 23rd of March.
4. **quantityDiscountQualifier(order: Order): Boolean**
    - **Description:** Checks if the quantity of the order is more than 5 units.
5. **salesThroughAppQualifier(order: Order): Boolean**
    - **Description:** Checks if the sale was made through the app.
6. **visaCardQualifier(order: Order): Boolean**
    - **Description:** Checks if the payment method is a Visa card.

3. Calculation Rule Functions:

- **Functionality:** Calculate discounts based on specific criteria.

1. **expiringDiscount(order: Order): Double**
    - **Description:** Calculates discount based on the remaining days for product expiry.

2. **cheeseAndWineDiscountCalculator(order: Order): Double**
    - **Description:** Calculates discount for cheese and wine products.
3. **specialDiscountCalculator(order: Order): Double**
    - **Description:** Calculates special discount for orders made on 23rd of March.
4. **quantityDiscountCalculator(order: Order): Double**
    - **Description:** Calculates discount based on the quantity of the order.
5. **salesThroughAppDiscountCalculator(order: Order): Double**
    - **Description:** Calculates discount for orders made through the app.
6. **visaCardDiscountCalculator(order: Order): Double**
    - **Description:** Calculates discount for orders made using Visa cards.

4. getOrderWithDiscount(order: Order, roleList: List[(Qualifier, Calculator)]): String

- **Functionality:** Process orders and return the order data with or without discount.
- **Description:** Determines the applicable discounts for an order based on the provided list of qualifying rules and calculation rules, calculates the final price, and returns the order data formatted as a string.

5. writeToDatabaseBatch(orderDataList: List[String]): Unit

- **Functionality:** Write data to the database in batch mode.
- **Description:** Writes the processed order data to a database table in batch mode, handling the database connection, table creation (if not exists), and batch insertion of order data.

6. Aggregated Logs (totalOrdersRead: Int, ordersInserted: Int, ordersWithDiscount: Int, ordersWithoutDiscount: Int): Unit

- **Functionality:** Log counts related to order processing.
- **Description:** Logs various counts related to the order processing, such as total orders read, orders inserted into the database, orders with and without discounts, using the provided logger instance.