**Lab Exercise 19- Manipulating Facts in Drools**

In Drools, **manipulating facts** refers to the process of inserting, modifying, or retracting facts (objects) in the working memory during the execution of rules. Facts are the data against which rules are evaluated, and manipulating them dynamically can be essential for complex decision-making processes.

**Objective:**

* Learn how to manipulate facts (insert, update, and retract) in Drools rules.
* Use the insert(), update(), and retract() methods in the Drools rule engine.
* Implement a scenario where we modify facts in the working memory during rule execution.

**Scenario:**

In this lab exercise, we'll simulate an e-commerce system where we check an order's total amount, apply a discount, and manage inventory. We will manipulate facts during rule execution by:

* **Inserting** new facts (e.g., apply a discount).
* **Updating** facts (e.g., mark an order as processed).
* **Retracting** facts (e.g., remove facts from working memory when they are no longer relevant).

**Step 1: Define Java Model Classes**

**Order.java**

This class represents an order, including the total amount and whether the order has been processed.

package com.example.model;

public class Order {

private String id;

private double totalAmount;

private boolean processed;

public Order(String id, double totalAmount) {

this.id = id;

this.totalAmount = totalAmount;

this.processed = false;

}

public String getId() {

return id;

}

public double getTotalAmount() {

return totalAmount;

}

public void setTotalAmount(double totalAmount) {

this.totalAmount = totalAmount;

}

public boolean isProcessed() {

return processed;

}

public void setProcessed(boolean processed) {

this.processed = processed;

}

@Override

public String toString() {

return "Order{id='" + id + "', totalAmount=" + totalAmount + ", processed=" + processed + '}';

}

}

**Discount.java**

This class represents a discount applied to an order.

package com.example.model;

public class Discount {

private String orderId;

private double discountAmount;

public Discount(String orderId, double discountAmount) {

this.orderId = orderId;

this.discountAmount = discountAmount;

}

public String getOrderId() {

return orderId;

}

public double getDiscountAmount() {

return discountAmount;

}

@Override

public String toString() {

return "Discount{orderId='" + orderId + "', discountAmount=" + discountAmount + '}';

}

}

**Step 2: Define Drools Rules (factManipulationRules.drl)**

In the rule file, we will manipulate facts using the insert(), update(), and retract() functions.

package com.example.rules;

import com.example.model.Order;

import com.example.model.Discount;

// Rule to apply a discount to orders with a total amount >= 100

rule "Apply Discount to High Value Orders"

when

$order : Order(totalAmount >= 100, processed == false)

then

Discount discount = new Discount($order.getId(), 10.0);

insert(discount); // Insert a new discount fact

System.out.println("Discount applied to order " + $order.getId());

end

// Rule to process the order after applying the discount

rule "Process Order"

when

$order : Order(processed == false)

$discount : Discount(orderId == $order.getId())

then

$order.setProcessed(true); // Mark the order as processed

update($order); // Update the order fact in the session

System.out.println("Order " + $order.getId() + " is now processed.");

end

// Rule to remove discount from the session after processing

rule "Remove Discount After Processing"

when

$discount : Discount()

$order : Order(id == $discount.getOrderId(), processed == true)

then

retract($discount); // Remove the discount fact from the session

System.out.println("Discount for order " + $order.getId() + " retracted.");

end

**Step 3: Define kmodule.xml**

The kmodule.xml file configures the knowledge base and session settings.

<?xml version="1.0" encoding="UTF-8"?>

<kmodule xmlns="http://jboss.org/kie/6.0.0/kmodule">

<kbase name="factManipulationKBase" packages="com.example.rules">

<ksession name="factManipulationKSession" type="stateful"/>

</kbase>

</kmodule>

**Step 4: Implement the Test Class (FactManipulationTest.java)**

In the test class, we will create an order and execute the rules in a stateful session to observe how facts are manipulated in the working memory.

package com.example.model;

import org.kie.api.KieServices;

import org.kie.api.runtime.KieContainer;

import org.kie.api.runtime.KieSession;

public class FactManipulationTest {

public static void main(String[] args) {

// Load KieServices and KieContainer

KieServices ks = KieServices.Factory.get();

KieContainer kContainer = ks.getKieClasspathContainer();

// Create a stateful Kie session

KieSession kSession = kContainer.newKieSession("factManipulationKSession");

// Create a new order with a total amount of 150

Order order = new Order("Order-001", 150);

kSession.insert(order); // Insert the order into the session

// Fire all rules

kSession.fireAllRules();

// Dispose of the session

kSession.dispose();

}

}

**Explanation of the Rules:**

1. **Insert a Discount**:
   * If the order's total amount is greater than or equal to 100 and the order is not yet processed, a new Discount object is created and inserted into the session using insert().
2. **Update the Order**:
   * Once a discount is applied, the order is marked as processed, and the order fact is updated in the working memory using update().
3. **Retract the Discount**:
   * After the order has been processed, the discount fact is no longer needed, so it is removed from the session using retract().

**Step 5: Run the Application**

When you run the FactManipulationTest.java class, the rules will be fired in sequence, and you will see the manipulation of facts through insertion, updates, and retraction in the Drools working memory.

**Expected Output:**

Discount applied to order Order-001

Order Order-001 is now processed.

Discount for order Order-001 retracted.

**Key Points:**

* **Insert**: The insert() function is used to add a new fact (in this case, a Discount) into the working memory.
* **Update**: The update() function modifies an existing fact in the working memory, so rules that depend on the updated fact will re-evaluate.
* **Retract**: The retract() function removes a fact from the working memory, so it no longer participates in rule evaluations.

This lab exercise demonstrates how to manipulate facts in Drools using insert(), update(), and retract() to implement a dynamic rule-based system where facts change during rule execution.