**Lab Exercise 21- Manipulating Facts in Code vs. Manipulating Facts in Rules**

**Objective:**

Understand the differences between manipulating facts directly in code and within Drools rules by working through two scenarios where each method is applied.

**Scenario:**

You are managing a fleet of Vehicle objects in a logistics system. The system must decide which vehicles need servicing based on their mileage. You will implement two approaches:

1. **Manipulating facts in code:** Handle all updates to vehicle statuses before firing the rules.
2. **Manipulating facts in rules:** Let the Drools engine manage the updates within the rules themselves.

**Step 1: Set Up the Project**

Ensure your Maven project is set up with the necessary Drools dependencies. You can use your existing project setup or create a new one.

**Step 2: Define the Model**

Create a Vehicle class that will be used as a fact in the Drools rules.

package com.example.model;

public class Vehicle {

private String id;

private int mileage;

private String status;

public Vehicle(String id, int mileage, String status) {

this.id = id;

this.mileage = mileage;

this.status = status;

}

public String getId() {

return id;

}

public void setId(String id) {

this.id = id;

}

public int getMileage() {

return mileage;

}

public void setMileage(int mileage) {

this.mileage = mileage;

}

public String getStatus() {

return status;

}

public void setStatus(String status) {

this.status = status;

}

@Override

public String toString() {

return "Vehicle{" +

"id='" + id + '\'' +

", mileage=" + mileage +

", status='" + status + '\'' +

'}';

}

}

**Part 1: Manipulating Facts in Code**

**Step 3: Create the Rule File**

Create a DRL file that will apply rules to Vehicle objects without manipulating facts.

package com.example.rules;

import com.example.model.Vehicle;

rule "Check Vehicle Status"

when

$vehicle : Vehicle( status == "Needs Service" )

then

System.out.println($vehicle.getId() + " requires servicing.");

end

**Step 4: Write the Main Class for Code-Based Manipulation**

In this part, you will handle fact manipulation (e.g., updating the status of a Vehicle) directly in the code before firing the rules.

package com.example.model;

import com.example.model.Vehicle;

import org.kie.api.KieServices;

import org.kie.api.runtime.KieContainer;

import org.kie.api.runtime.KieSession;

public class DroolsTest {

public static void main(String[] args) {

// Load the knowledge base

KieServices ks = KieServices.Factory.get();

KieContainer kContainer = ks.getKieClasspathContainer();

KieSession kSession = kContainer.newKieSession("ksession-rules");

// Create Vehicle objects

Vehicle vehicle1 = new Vehicle("V001", 12000, "Active");

Vehicle vehicle2 = new Vehicle("V002", 8000, "Active");

Vehicle vehicle3 = new Vehicle("V003", 16000, "Active");

// Manually update status based on mileage

if (vehicle1.getMileage() > 10000) {

vehicle1.setStatus("Needs Service");

}

if (vehicle2.getMileage() > 10000) {

vehicle2.setStatus("Needs Service");

}

if (vehicle3.getMileage() > 10000) {

vehicle3.setStatus("Needs Service");

}

// Insert Vehicle objects into the session

kSession.insert(vehicle1);

kSession.insert(vehicle2);

kSession.insert(vehicle3);

// Fire all rules

kSession.fireAllRules();

// Dispose the session

kSession.dispose();

}

}

**Part 2: Manipulating Facts in Rules**

**Step 5: Create the Rule File with Fact Manipulation**

Now, modify the DRL file to update the status of Vehicle objects directly within the rules.

package com.example.rules;

import com.example.model.Vehicle;

rule "Update Vehicle Status"

when

$vehicle : Vehicle( mileage > 10000, status == "Active" )

then

$vehicle.setStatus("Needs Service");

update($vehicle);

System.out.println($vehicle.getId() + " status updated to 'Needs Service'.");

end

rule "Check Vehicle Status"

when

$vehicle : Vehicle( status == "Needs Service" )

then

System.out.println($vehicle.getId() + " requires servicing.");

end

**Step 6: Write the Main Class for Rule-Based Manipulation**

Here, you will rely on Drools rules to manage the status updates of the Vehicle objects.

package com.example.model;

import com.example.model.Vehicle;

import org.kie.api.KieServices;

import org.kie.api.runtime.KieContainer;

import org.kie.api.runtime.KieSession;

public class DroolsTest {

public static void main(String[] args) {

// Load the knowledge base

KieServices ks = KieServices.Factory.get();

KieContainer kContainer = ks.getKieClasspathContainer();

KieSession kSession = kContainer.newKieSession("ksession-rules");

// Create Vehicle objects

Vehicle vehicle1 = new Vehicle("V001", 12000, "Active");

Vehicle vehicle2 = new Vehicle("V002", 8000, "Active");

Vehicle vehicle3 = new Vehicle("V003", 16000, "Active");

// Insert Vehicle objects into the session

kSession.insert(vehicle1);

kSession.insert(vehicle2);

kSession.insert(vehicle3);

// Fire all rules

kSession.fireAllRules();

// Dispose the session

kSession.dispose();

}

}

**Step 7: Run Both Parts and Compare Results**

* **Part 1 (Code-Based Manipulation):** Run the first DroolsTest class. You will see the output where the status of each vehicle is manually updated before the rules are fired.
* **Part 2 (Rule-Based Manipulation):** Run the second DroolsTest class. The output will show how the Drools engine dynamically updates the vehicle status and determines which vehicles need servicing.

**Expected Output Comparison:**

**Part 1 (Code-Based Manipulation):**

V001 requires servicing.

V003 requires servicing.

**Part 2 (Rule-Based Manipulation):**

V001 status updated to 'Needs Service'.

V001 requires servicing.

V003 status updated to 'Needs Service'.

V003 requires servicing.

**Discussion:**

* **Part 1 (Code-Based Manipulation):** All fact manipulations are handled in the code, giving you full control over the process. However, it requires more boilerplate and explicit handling.
* **Part 2 (Rule-Based Manipulation):** The Drools engine takes care of fact manipulation, leading to more dynamic and flexible rule execution. It allows you to encapsulate business logic within rules but may be harder to trace/debug due to the abstraction.

**Deliverables:**

* Two separate Java projects or classes demonstrating code-based and rule-based fact manipulation.
* A report comparing the results and discussing the advantages and disadvantages of each approach.

This exercise should give you a clear understanding of when and why you might choose to manipulate facts in code versus within rules, and how each method impacts the design and flexibility of your system.