**Lab Exercise 17- Timer Attribute in Drools**

In this lab, we’ll create a scenario where we simulate a system that checks the order status at regular intervals to see if an order has been processed. If the order is not processed within a certain period, it will trigger an alert.

**Objective:**

* Learn how to use the **Timer** attribute in Drools.
* Simulate periodic rule execution using a **stateful session**.
* Use **agenda-group** and **timer** in Drools rules to execute rules at intervals.

**Scenario:**

In this lab, we will simulate an order processing system where we check the status of an order periodically (e.g., every 3 seconds) until the order is processed. After a certain delay (e.g., 10 seconds), if the order is still not processed, we will send an alert.

**Project Structure:**

src

├── main

│ ├── java

│ │ └── com.example.model

│ │ ├── Order.java

│ │ └── OrderProcessingTimerTest.java

│ └── resources

│ ├── META-INF

│ │ └── kmodule.xml

│ └── rules

│ └── orderProcessingTimerRules.drl

**Step 1: Define the Java Model Class (Order.java)**

package com.example.model;

public class Order {

private String id;

private boolean processed;

public Order(String id) {

this.id = id;

this.processed = false;

}

public String getId() {

return id;

}

public boolean isProcessed() {

return processed;

}

public void setProcessed(boolean processed) {

this.processed = processed;

}

@Override

public String toString() {

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

}

}

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

In this rule file, we will use the **Timer** attribute to check the order status every 3 seconds. If the order is not processed within 10 seconds, we will trigger an alert.

package com.example.rules;

import com.example.model.Order;

// This rule checks the status of the order every 3 seconds

rule "Check Order Status"

timer(int: 3s)

when

$order : Order(processed == false)

then

System.out.println("[CHECK] Order " + $order.getId() + " is still not processed.");

end

rule "Check Order process Status"

timer(int: 3s)

when

$order : Order(processed == true)

then

System.out.println("[CHECK] Order " + $order.getId() + " is processed.");

end

// This rule triggers after 10 seconds if the order is not processed

rule "Alert Unprocessed Order"

timer(int: 10s)

when

$order : Order(processed == false)

then

System.out.println("[ALERT] Order " + $order.getId() + " is still not processed after 10 seconds!");

end

**Step 3: Define kmodule.xml**

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

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

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

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

</kbase>

</kmodule>

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

This class will initialize a **stateful** session and insert an Order object. The session will keep firing rules periodically due to the timer attribute.

package com.example.model;

import org.kie.api.KieServices;

import org.kie.api.runtime.KieContainer;

import org.kie.api.runtime.KieSession;

public class OrderProcessingTimerTest {

public static void main(String[] args) throws InterruptedException {

// Load KieServices and KieContainer

KieServices ks = KieServices.Factory.get();

KieContainer kContainer = ks.getKieClasspathContainer();

// Create a stateful Kie session

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

// Create a sample order and insert it into the session

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

kSession.insert(order);

// Start a new thread for firing the rules asynchronously

new Thread(kSession::fireUntilHalt).start();

// Simulate order processing delay

Thread.sleep(5000); // After 5 seconds, mark the order as processed

order.setProcessed(true);

kSession.update(kSession.getFactHandle(order), order);

// Allow some time for rules to fire before shutting down

Thread.sleep(10000); // Wait for 10 seconds

// Stop the rule engine

kSession.halt();

kSession.dispose();

}

}

**Explanation of the Test:**

1. **Timer Rules**:
   * Check Order Status: This rule fires every 3 seconds and checks if the order has been processed.
   * Alert Unprocessed Order: This rule triggers after 10 seconds if the order is still not processed.
2. **Stateful Session**:
   * We use a **stateful** session to keep the order in working memory and to execute the rules periodically using the fireUntilHalt() method.
   * The session runs in a separate thread, continuously firing rules as long as the session is running.
3. **Simulated Delay**:
   * After inserting the order, the system waits for 5 seconds before marking the order as processed. This gives enough time for the first rule (Check Order Status) to fire, but not enough time for the second rule (Alert Unprocessed Order) to trigger an alert.

**Step 5: Run the Application**

When you run the OrderProcessingTimerTest.java class, you should see the periodic execution of rules every 3 seconds until the order is processed. If the order is not processed after 10 seconds, the alert will be triggered.

**Expected Output:**

[CHECK] Order Order-001 is still not processed.

[CHECK] Order Order-001 is still not processed.

[CHECK] Order Order-001 is still not processed.

Order processed after 5 seconds.

If you delay the processing of the order, the output could include the alert as well:

[CHECK] Order Order-001 is still not processed.

[CHECK] Order Order-001 is still not processed.

[ALERT] Order Order-001 is still not processed after 10 seconds!

**Key Points:**

* **Timer Attribute**: The timer(int: Xs) attribute is used to specify the interval between rule executions. In this example, we execute rules every 3 seconds and trigger an alert if the order is unprocessed after 10 seconds.
* **Stateful Session**: A stateful session allows the rule engine to maintain the state of the order in memory, allowing the rules to continuously check the order status until the session is halted.

This lab exercise demonstrates how to use the **Timer** attribute in Drools for periodic rule execution within a stateful session.