**Backword-Chaining in Drools**

**Overview:**

In Drools, backward chaining is a reasoning approach where the system tries to deduce facts by working backward from a goal (conclusion) to find the facts (conditions) that support it. This contrasts with forward chaining, where rules are applied as soon as their conditions are satisfied.

In Drools, backward chaining can be enabled using **queries**. A query is a special type of rule that allows you to ask questions about the current state of the working memory and deduce facts dynamically. The system will attempt to satisfy the query by deriving new facts or finding existing ones in working memory.

**Example of Backward Chaining in Drools**

Here's an example of how you can implement backward chaining using a query.

**1. Define a Query in rules.drl:**

package com.example.rules;

import com.example.TemperatureSensor;

query "Is High Temperature"

$sensor : TemperatureSensor(temperature > 75)

end

rule "Alert High Temperature"

when

// The query "Is High Temperature" is invoked to check if the condition is met

(Is High Temperature( $sensor ; ) )

then

System.out.println("ALERT: High temperature detected! Sensor ID: " + $sensor.getSensorId());

end

**2. Example Application Using the Query:**

In the application, you can use the query to check whether a fact that satisfies the query exists, or derive it dynamically:

package com.example;

import org.kie.api.KieServices;

import org.kie.api.runtime.KieContainer;

import org.kie.api.runtime.KieSession;

import org.kie.api.runtime.rule.QueryResults;

import org.kie.api.runtime.rule.QueryResultsRow;

public class TemperatureMonitoringApp {

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 a TemperatureSensor object

TemperatureSensor sensor = new TemperatureSensor("Sensor-1", 80.0);

// Insert the object into the session

kSession.insert(sensor);

// Fire all rules

kSession.fireAllRules();

// Perform backward chaining by querying for high temperatures

QueryResults results = kSession.getQueryResults("Is High Temperature");

for (QueryResultsRow row : results) {

TemperatureSensor s = (TemperatureSensor) row.get("$sensor");

System.out.println("Query result: Sensor " + s.getSensorId() + " has high temperature.");

}

// Dispose of the session

kSession.dispose();

}

}

**Key Points:**

* **Query**: Defines a condition that Drools will try to satisfy by searching or deriving facts.
* **Rule referencing the query**: The rule "Alert High Temperature" uses the result of the query "Is High Temperature" to fire only when the condition is met.
* **Query Execution**: The application explicitly executes the query using kSession.getQueryResults() to perform backward chaining and check if any TemperatureSensor with a temperature above 75 exists.

This approach allows backward chaining to dynamically search for facts that satisfy the query condition.