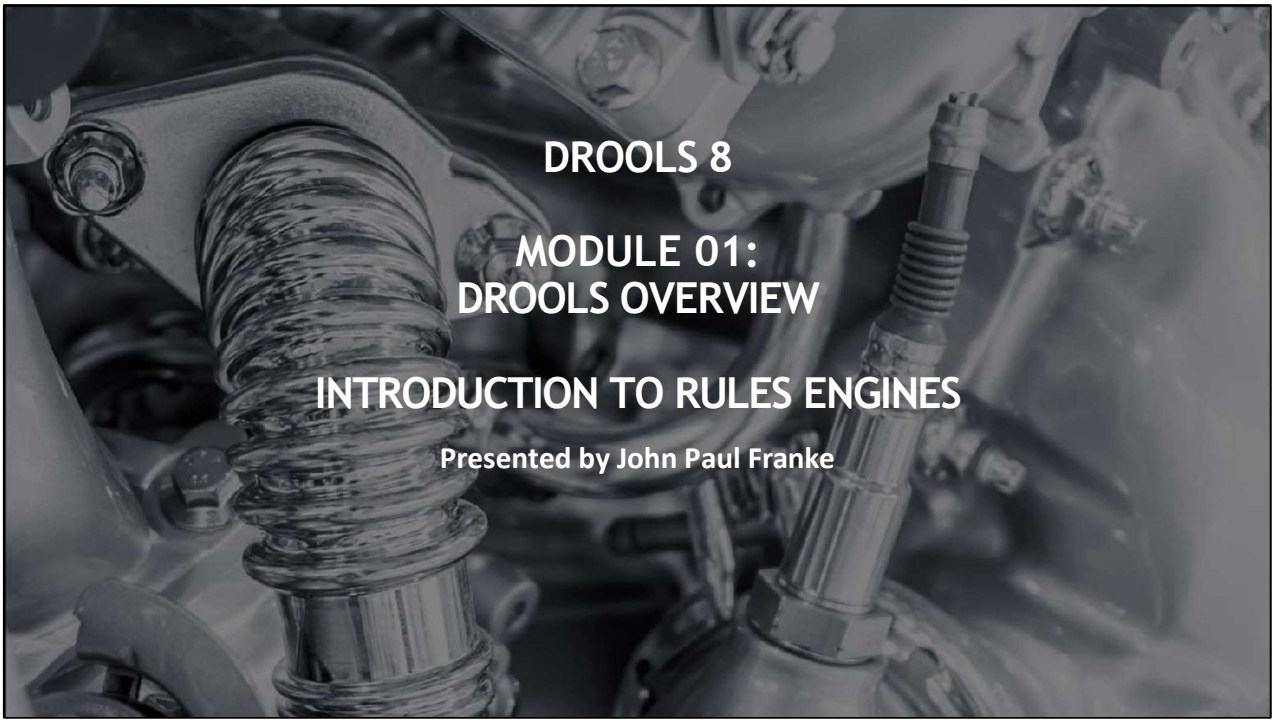
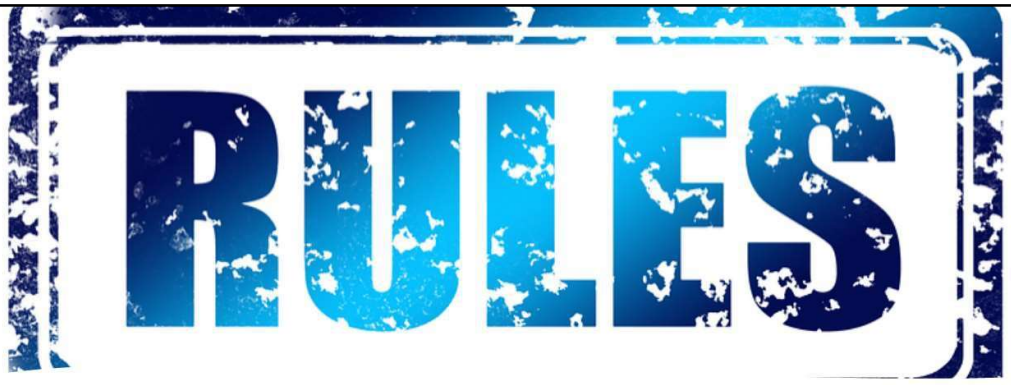




Course Overview

Welcome to our comprehensive course on Rule Engines, with a special focus on Drools, one of the most popular and powerful rule engines available today. This course is designed to equip you with both theoretical knowledge and practical skills in the realm of rule-based systems. Here's what you can expect:

Course Objective:

Our main goal is to help you understand the concept of rule engines, why they are used, and how to effectively implement them using Drools. By the end of this course, you'll be well-versed in the fundamentals of rule engines and proficient in using Drools for automating complex decision-making processes.

Learning Objectives

Here are our learning goals for this Module:

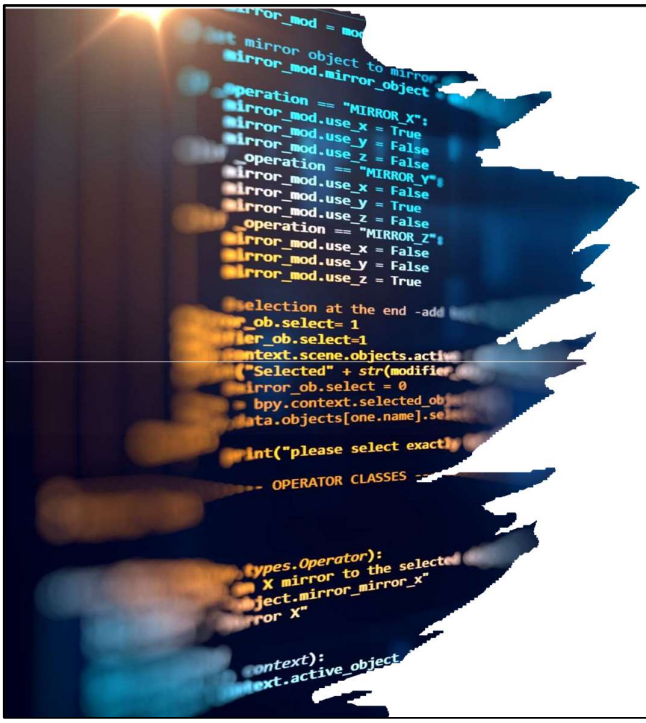
1. Understanding Rule Engines
2. Historical Context and Evolution of Rule Engines
3. Procedural vs. Declarative Programming
4. Case Studies: Appropriate Use Cases for Rule Engines
5. An Introduction to the Drools Platform
6. The Lifecycle of Rule Execution
7. Basics of Rule Anatomy in Drools
8. Interactive Quiz: Assessing Understanding

Understanding Rule Engines

Imagine you're a chef in a big, busy kitchen. You have a recipe book (the rule engine) that tells you how to make different dishes (business decisions). Each recipe (rule) in your book gives specific instructions on what to do if certain ingredients (data) are available. For example, if you have eggs, flour, and sugar, the recipe tells you to make a cake.

Understanding Rule Engines

In a professional kitchen with lots of ingredients coming in and out, you need to make decisions quickly. This is where the rule engine really shines. It's like having an assistant chef who knows all the recipes by heart and can instantly tell you which dish you can make with the ingredients at hand. You don't have to flip through the book yourself; the assistant does it for you, saving time and making your kitchen more efficient.

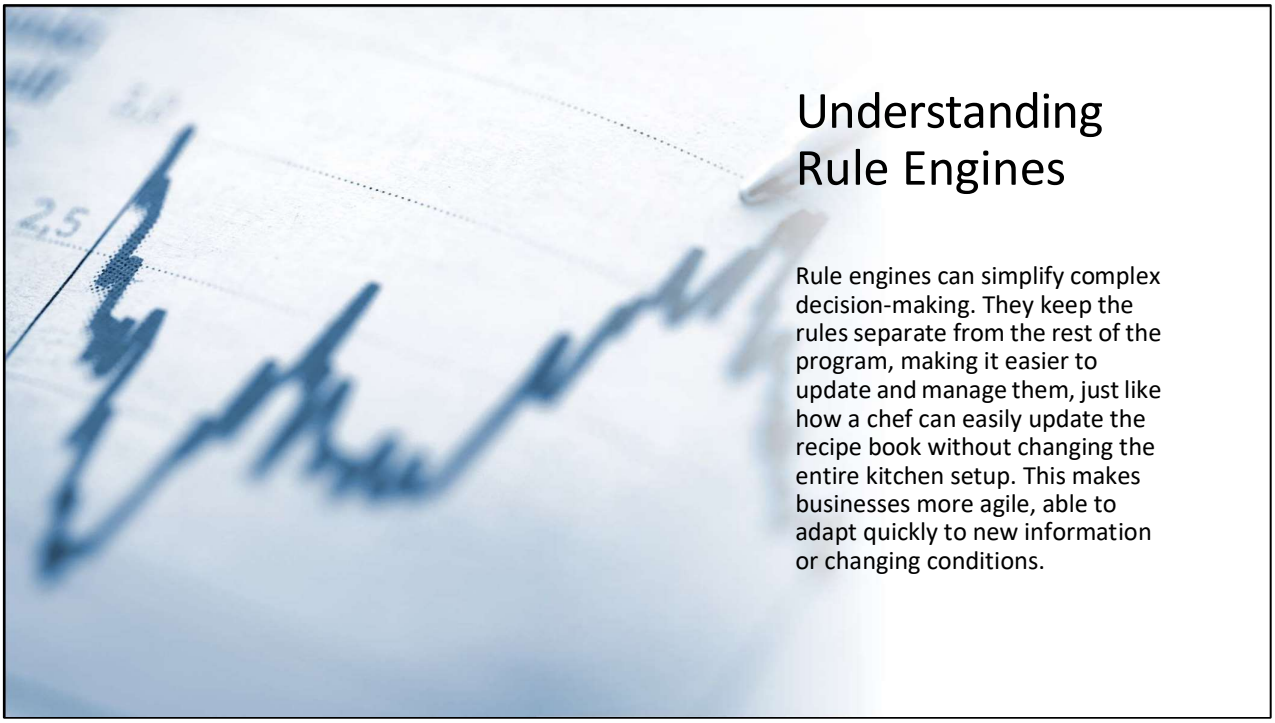


Understanding Rule Engines

In the world of computer programming, a Rule Engine is a software system that helps make decisions based on a set of rules. These rules are like recipes, but instead of food ingredients, they use data and information to make decisions.

Understanding Rule Engines

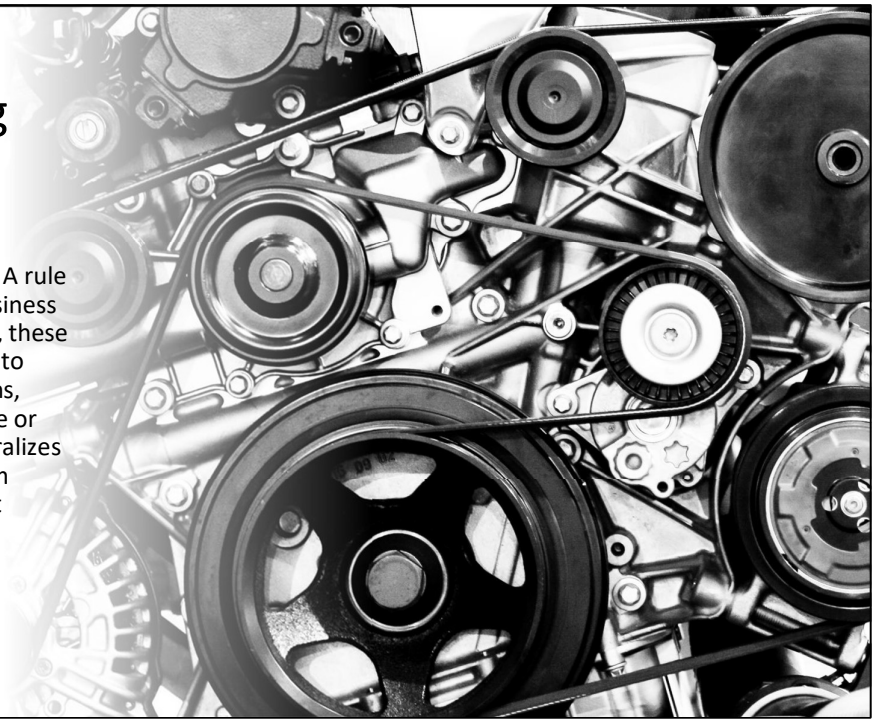
For example, in a banking application, a rule might say, "If a customer has savings over $10,000 and a good credit score, offer them a premium credit card." The rule engine looks at the customer data (like the chef looks at ingredients) and decides what offer to make.



Understanding Rule Engines

Business Rules: These are like the guidelines or conditions that determine how a business operates. For example, a rule in a retail store might be, "If a customer buys more than $100 worth of goods, they get a 10% discount." These rules can be about anything related to the business process.

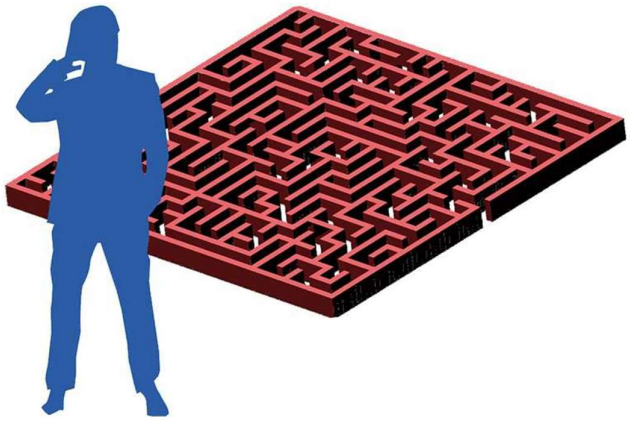
Understanding Rule Engines



Automation and Efficiency: A rule engine automates these business rules. Without a rule engine, these rules might be hardcoded into various software applications, making them hard to change or manage. A rule engine centralizes these rules and applies them consistently across different applications and systems.

Understanding Rule Engines

Complex Logic Handling: Rule engines are particularly useful in scenarios where there is complex decision logic involving multiple business rules. They can efficiently process large sets of conditions and rules to provide accurate outcomes.

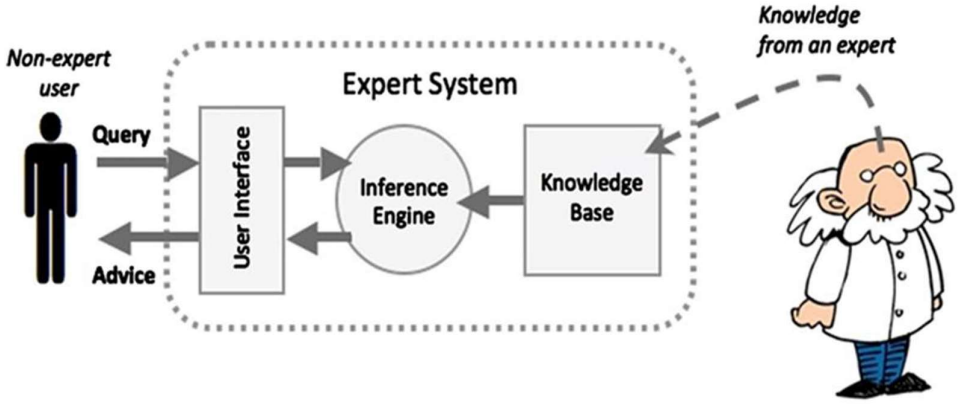


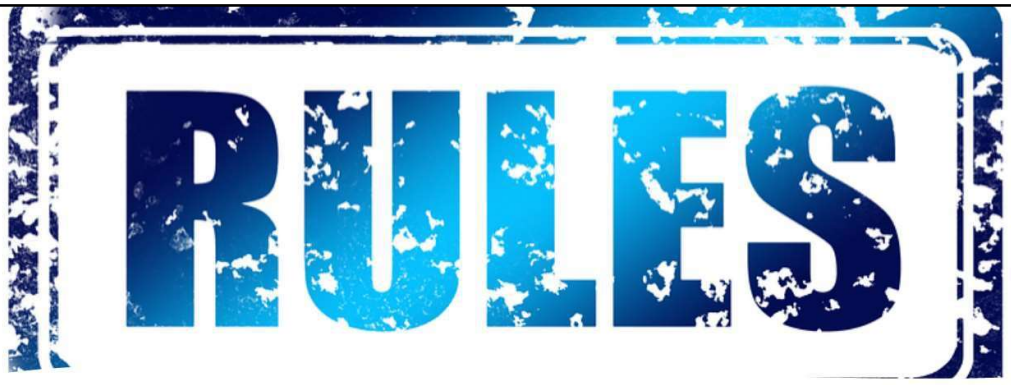
History & Evolution of R.E.'s

Domain Experts Era: In the early days of complex decision-making in business and technology, organizations heavily relied on domain experts. These were individuals with extensive knowledge and experience in specific areas, such as finance, healthcare, or engineering. They made decisions based on their expertise, intuition, and understanding of best practices. However, relying solely on human experts had its limitations, including inconsistency in decision-making and challenges in handling large amounts of data.

History & Evolution of R.E.'s

Rise of Expert Systems: To address these limitations, the concept of expert systems emerged in the 1970s and 1980s. An expert system is a type of artificial intelligence program that emulates the decision-making ability of a human expert. It uses a knowledge base of human expertise and an inference engine that applies rules to the knowledge base to provide advice or make decisions. The idea was to capture the knowledge of domain experts in a systematic and structured way. This allowed for more consistent and scalable decision-making.



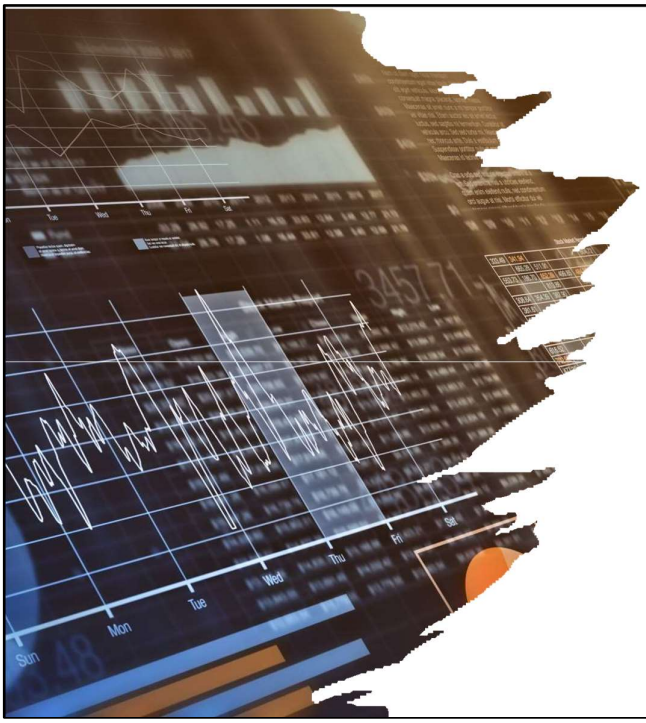


History & Evolution of R.E.'s

Development of Rule-Based Systems: Expert systems evolved into rule-based systems, where the focus shifted to the rules themselves. In these systems, the knowledge of the expert was encoded in the form of if-then rules. For example, “If a customer spends more than $100, then they receive a 10% discount.” These rules could be written and maintained by business analysts, reducing the dependency on IT teams for every change or update.

History & Evolution of R.E.'s

Evolution into Rule Engines: Rule engines emerged as a more advanced and flexible version of rule-based systems. Unlike the earlier systems, rule engines separated the rule processing from the application code. This meant that rules could be changed without altering the underlying software, providing greater agility and adaptability to changing business environments. Rule engines also offered enhanced capabilities like handling complex, interrelated rules, and performing high-speed processing.



History & Evolution of R.E.'s

Current Trends and Future: Today, rule engines are often part of larger business automation platforms, incorporating big data, machine learning, and cloud computing. This allows them to process vast amounts of data, learn from historical decisions, and adapt rules dynamically, making them even more powerful and insightful.

Advantages and Limits

- **Complex Decision Logic**: If your application involves complex decision-making, a rule engine can manage this complexity more efficiently than hard-coded logic.

Advantages and Limits

- **Frequent Rule Changes**: In environments where business rules change often, rule engines allow for quick updates without needing to modify and redeploy the application code.

Advantages and Limits

- **Need for Business User Control**:
  If business analysts or policy managers need to create or modify rules without involving IT, rule engines provide a user-friendly interface

Advantages and Limits

- **Auditing and Compliance**:
  Rule engines can provide detailed logs of decision-making processes, which is beneficial for auditing and regulatory co mpliance.

Advantages and Limits

When Not to Use Rule Engines

- **Simple Decision Logic**: If your application's decision logic is straightforward and unlikely to change, the overhead of implementing a rule engine might not be justified.

Advantages and Limits

When Not to Use Rule Engines

- **Performance Constraints**: Rule engines can introduce some processing overhead. In high-performance or real-time systems where every millisecond counts, directly coding the logic might be more efficient.

Advantages and Limits

When Not to Use Rule Engines

- **Limited Resources or Expertise**: Setting up and maintaining a rule engine requires certain expertise. If your team lacks this expertise or the resources to acquire it, it might be more pragmatic to stick with traditional coding approaches.

Advantages and Limits

When Not to Use Rule Engines

- **Small-Scale Applications**: For small-scale applications with a limited scope and lifecycle, the complexity and maintenance of a rule engine may not be necessary.

Procedural vs Declarative Programming

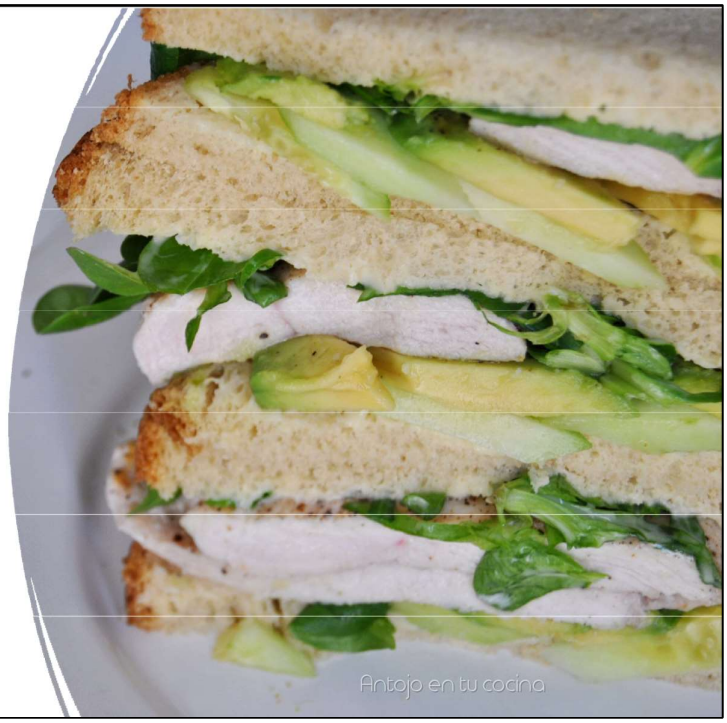
Procedural Programming:

Let's understand procedural vs. declarative programming using a simple, everyday analogy: preparing a meal:

Imagine you have a recipe for making a sandwich. This recipe lists each step you need to follow in order:

- Take two slices of bread.
- Spread butter on one slice.
- Add a layer of cheese.
- Put slices of tomato on top of the cheese.
- Cover it with the second slice of bread

Procedural vs Declarative Programming



Procedural Programming: Like Following a Recipe Step-by-Step

In procedural programming, just like following this recipe, you tell the computer exactly what steps to take and in what order. You provide a specific set of instructions (algorithms) to achieve the desired outcome. Just as you follow the recipe step-by-step to make a sandwich, the computer follows your written code step-by-step to complete a task.

Procedural vs Declarative Programming

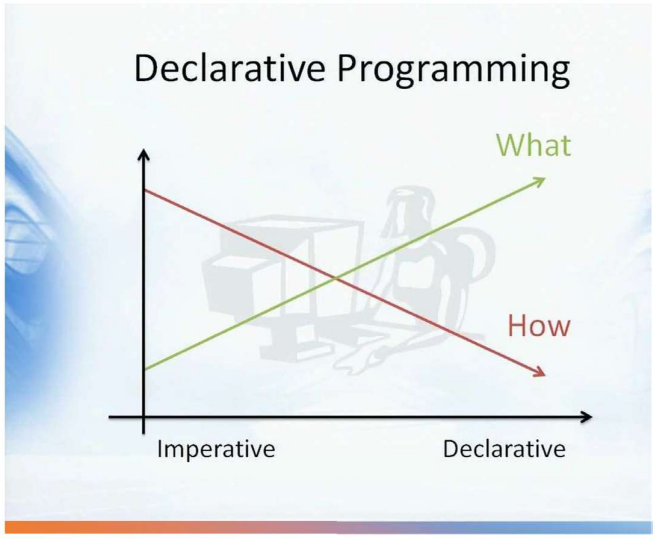
Declarative Programming:

Now, imagine instead of making the sandwich yourself, you go to a sandwich shop and simply ask for a "cheese and tomato sandwich." You don't provide step-by-step instructions on how to make it; you just declare what you want.

Procedural vs Declarative Programming

Key Differences:

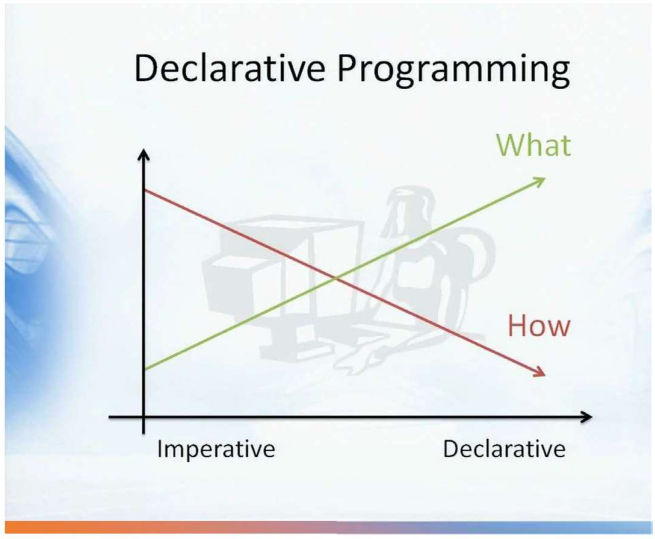
- **Control**: In procedural programming, you control the process, like following a recipe. In declarative programming, you control the outcome, like ordering a sandwich.



Procedural vs Declarative Programming

Key Differences:

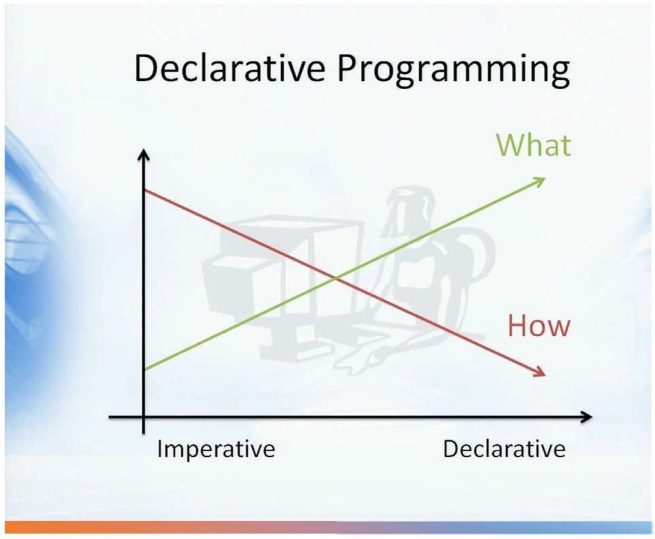
- **How vs. What**: Procedural focuses on 'how' to do things (the process), while declarative focuses on 'what' you want to achieve (the outcome).



Procedural vs Declarative Programming

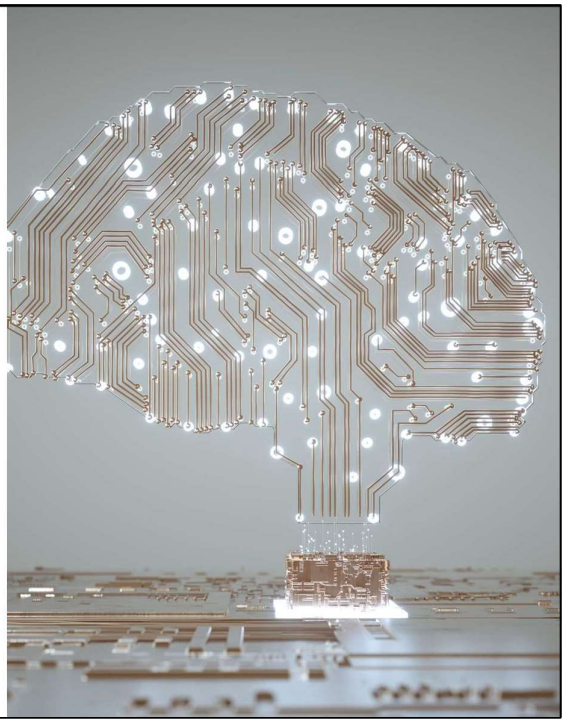
Key Differences:

- **Examples**:
  Procedural programming is seen in languages like C and Java. Declarative styles are used in SQL for database queries (you declare what data you want, not how to get it) and in HTML for web pages (you specify what elements you want, not how they should be rendered).



R.E's as early AI

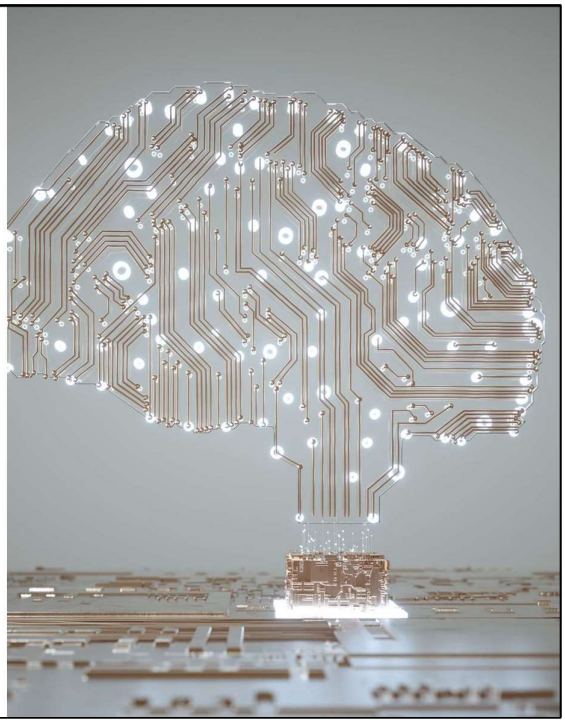
Rule engines are often considered an early form of Artificial Intelligence (AI) because of their ability to mimic certain aspects of human decision-making using predefined logic and rules.



R.E's as early AI

Mimicking Human Decision-Making

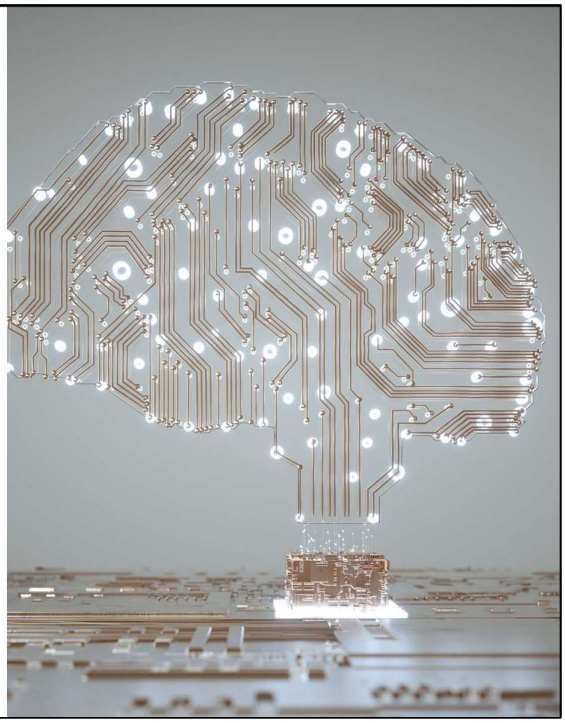
- **Expert Systems**: Expert systems were among the first successful forms of AI. These systems sought to emulate the decision-making ability of human experts. By encoding expert knowledge in the form of rules, these systems could make informed decisions, similar to how a human expert would.
- **Logic and Reasoning**: Like human reasoning, rule engines operate by applying logical rules to a given set of data. They evaluate conditions (if-then statements) and make decisions based on these evaluations, much like how we make decisions based on certain conditions or criteria in everyday life.



R.E's as early AI

Automated Decision Processes

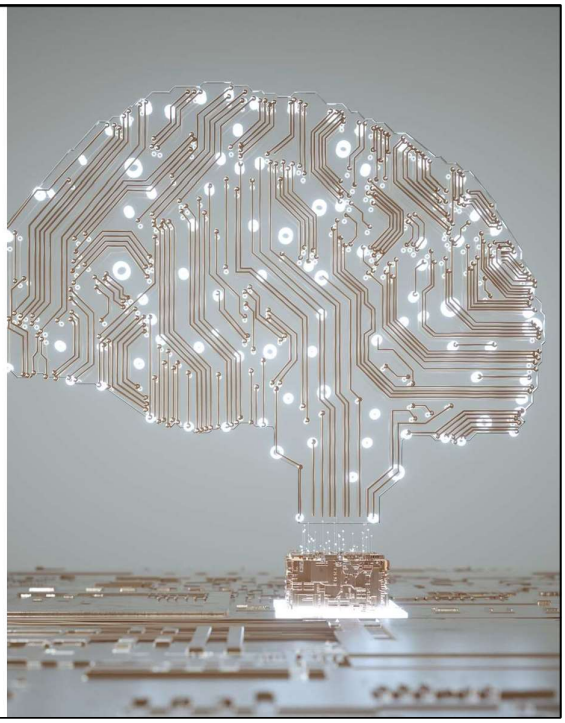
- **Processing Complex Rules**: Rule engines can handle complex, interconnected rules that might be challenging and time-consuming for humans to process manually. This capability to manage and reason through complex rule sets is a foundational aspect of AI.
- **Dynamic Decision Making**: Advanced rule engines can modify their behavior based on changing inputs, somewhat like learning. While not as sophisticated as modern machine learning algorithms, this dynamic response to changing data was an early step towards the adaptive behavior seen in AI.



R.E's as early AI

Scalability and Consistency

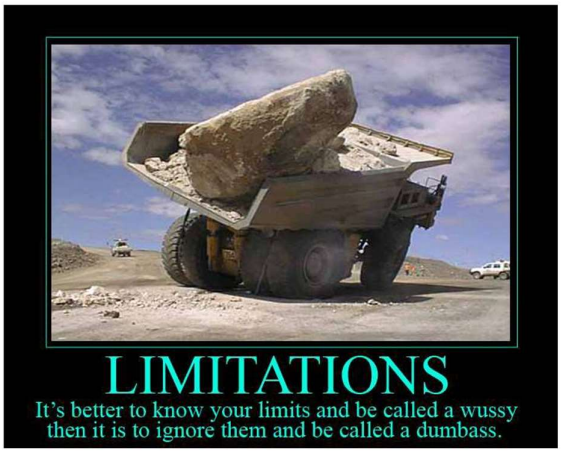
- **Handling Large Scale Data**: Rule engines can process vast amounts of data and make decisions much faster than a human could. This scalability is a key feature of AI, enabling efficient processing and analysis of large datasets.
- **Consistent Application of Rules**: Unlike humans, rule engines do not suffer from fatigue or inconsistency. They apply the same set of rules uniformly every time, ensuring consistent outcomes, a trait desirable in AI systems for reliable performance.



R.E's as early AI

Limitations Compared to Modern AI

- **Lack of Learning Ability**: Traditional rule engines do not have the ability to learn from past decisions or data patterns. They operate solely based on the rules provided to them. In contrast, modern AI, especially machine learning, continuously learns and improves from new data.
- **Rigid Rule Dependency**: Rule engines are entirely dependent on the rules they are given. They lack the intuitive understanding and flexibility that comes with more advanced AI, which can infer and adapt beyond hardcoded rules.



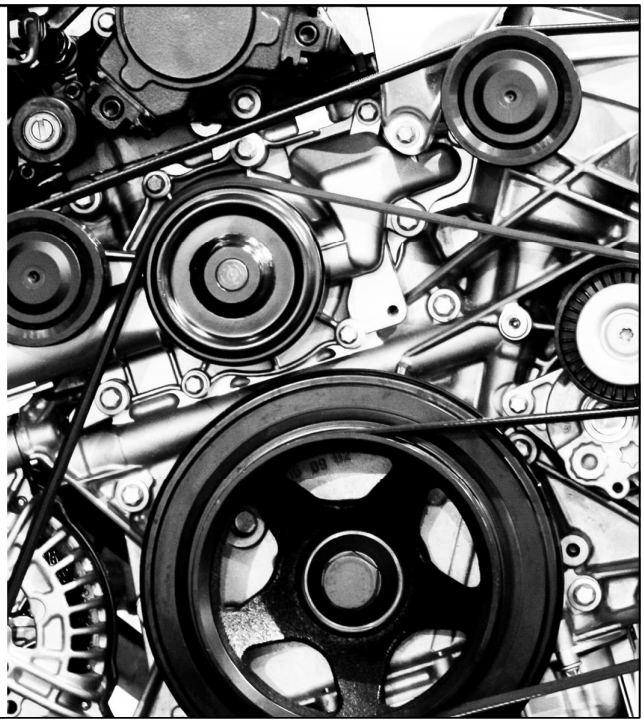
Rule Engine Components

To understand a rule engine better, let's break down its core components:

Rule Repository: This is like a library or database where all the rules are stored. Think of it as a cookbook containing different recipes. Each "recipe" (rule) in this repository defines certain conditions and the actions that should be taken when those conditions are met.

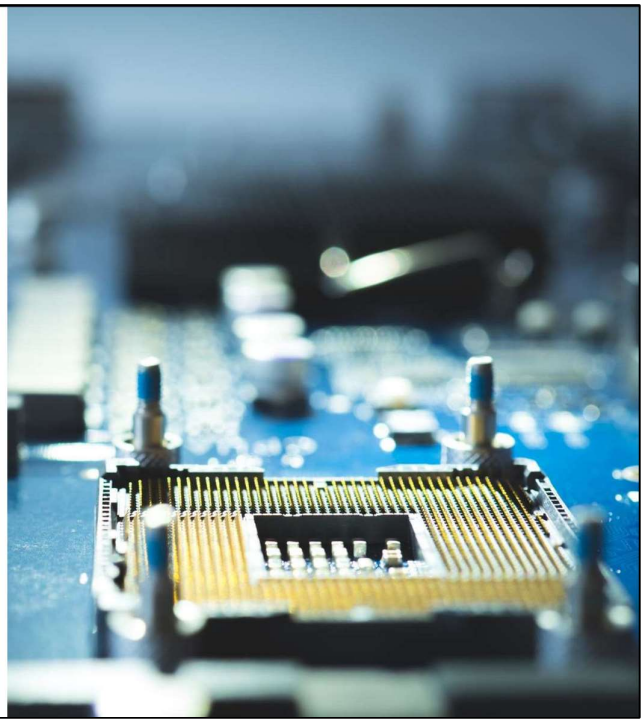
Rule Engine Components

Inference Engine: This is the heart of the rule engine. The inference engine takes your data, applies the rules from the repository, and then makes decisions or infers conclusions. It's like a chef who reads the recipes (rules) and decides what to cook based on the ingredients (data) available.

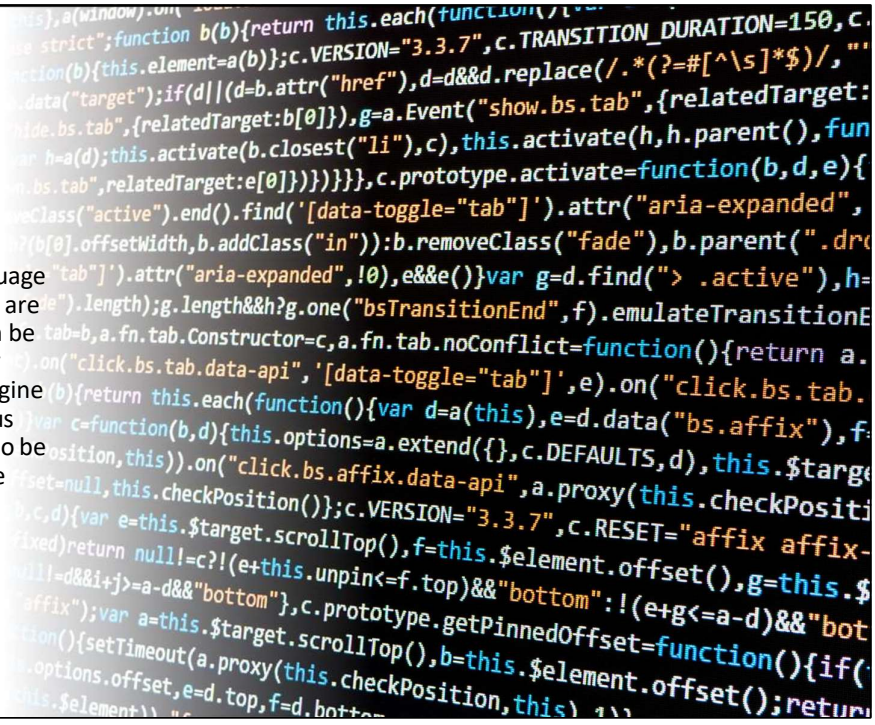


Rule Engine Components

Working Memory: In a rule engine, 'working memory' is a temporary storage area where all the data (known as 'facts') relevant to the rules are kept. This data can come from various sources, such as a database, user input, or other systems. Once in the working memory, this data is accessible and can be used by the rule engine to evaluate rules.



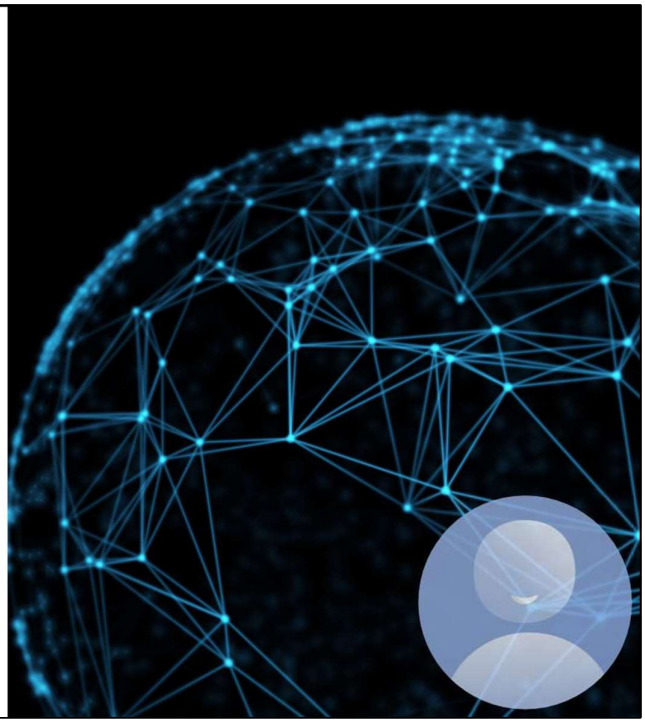
Rule Engine Components



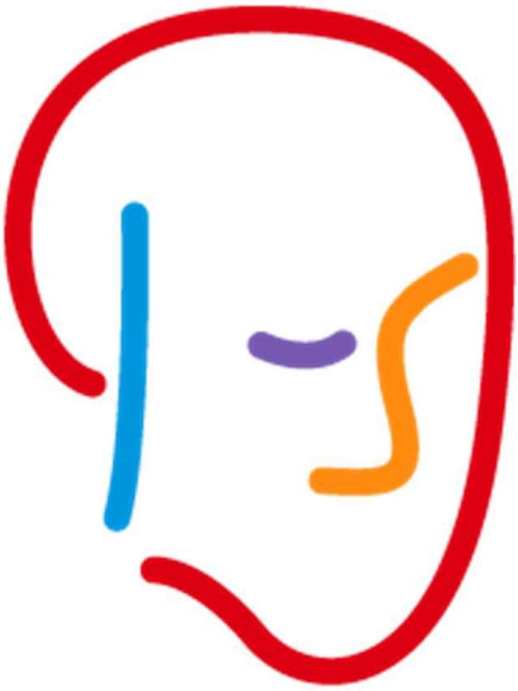
Rule Syntax: This is the language or format in which the rules are written. Just like recipes can be written in different styles or languages, rules in a rule engine can also be written in various syntaxes. The syntax needs to be understood by the inference engine.

Rule Engine Components

Rule Execution Environment: This provides the runtime environment for the rule engine. It includes the necessary infrastructure to load rules from the repository, execute them, and possibly integrate with other systems.



Introduction to Drools



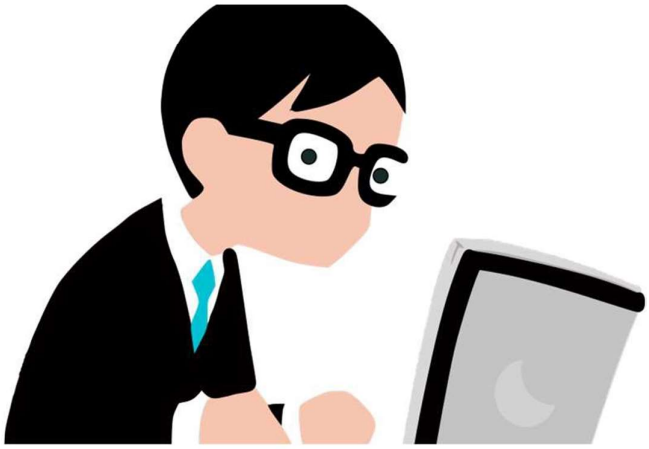
What is Drools?

Drools is an open-source Business Rules Management System (BRMS) that provides a robust framework for defining and executing business rules. It's like having a wise advisor in your software, helping make important decisions based on predefined logic.

Introduction to Drools

Drools was initially developed by Bob McWhirter, a software engineer and open-source enthusiast. Bob McWhirter began the Drools project in 2001 as an effort to create a flexible and robust rule engine based on the Rete algorithm, which is a popular method for pattern matching in artificial intelligence applications.

The project was named "Drools" as a play on the word "rules" and was developed as an open-source initiative from the start.

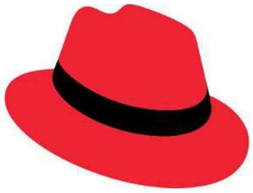


Introduction to Drools

Drools became part of the JBoss portfolio in 2005. This was a significant step in the evolution of Drools, as it marked the transition from being an independent open-source project to becoming a part of a larger and well-established suite of enterprise-level middleware solutions offered by JBoss. The integration into JBoss helped expand the visibility, development, and adoption of Drools, allowing it to benefit from the resources and community associated with JBoss.

Introduction to Drools

When Red Hat acquired JBoss in 2006, Drools came under the Red Hat umbrella. Under Red Hat, Drools continued to thrive as an open-source project, benefiting from Red Hat's resources and enterprise support while maintaining its community-driven development model.



Red Hat

The Rule Execution Lifecycle

Let's break down the Rule Execution Lifecycle into clear, easy-to-understand steps:

1. Rule Creation and Deployment

- **Authoring**: The lifecycle begins with the creation of rules. This involves defining the conditions (the 'if' part) and actions (the 'then' part) of each rule.
- **Compiling**: After rules are authored, they are compiled. This turns the human-readable rules into a format that the rule engine can process.
- **Deployment**: Compiled rules are then deployed to the rule engine. This is like adding the rules to the rule engine's 'rulebook' from which it can refer.

The Rule Execution Lifecycle

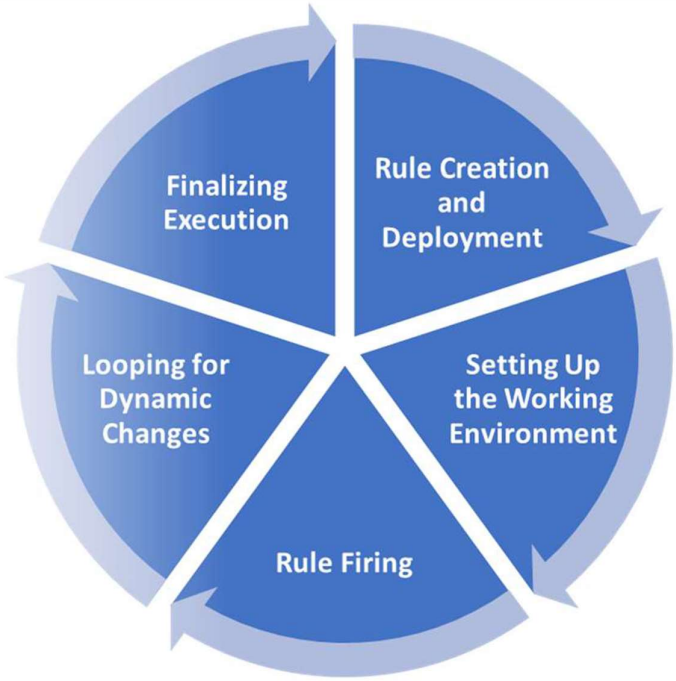
2. Setting Up the Working Environment

- **Initializing the Rule Engine**: Before executing any rules, the rule engine is initialized. This sets up the necessary environment for rule execution.
- **Defining a Knowledge Session**: Before rules can be fired, a 'knowledge session' or 'KIE session' is created. This is a crucial component that acts as a container for the rules, facts, and the interactions between them. The session container is linked to the working memory, ensuring that it has access to all the relevant facts needed for rule evaluation.
- **Loading Data (Facts)**: Data, or 'facts', relevant to the rules are loaded into the rule engine's working memory. These facts are the information the rule engine will use to evaluate against the rules.

The Rule Execution Lifecycle

3. Rule Firing

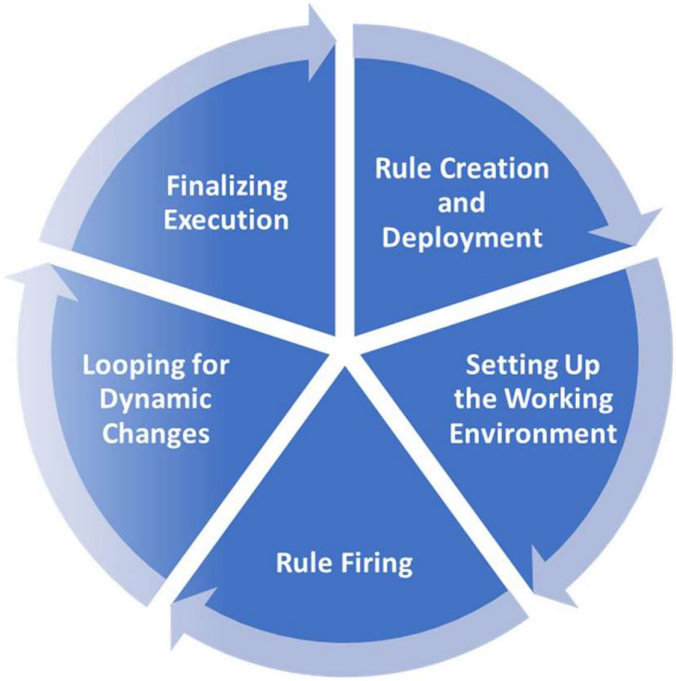
- **Agenda Creation**: The rule engine creates an 'agenda', which is a list of rules that are eligible to fire based on the current facts in the working memory.
- **Rule Evaluation**: The rule engine evaluates the facts against the conditions of each rule. When conditions of a rule are met, the rule is triggered or 'fired'.
- **Executing Actions**: For each fired rule, the corresponding actions are executed. This could be anything from calculating a value, making a decision, or modifying a fact in the working memory.



The Rule Execution Lifecycle

4. Looping for Dynamic Changes

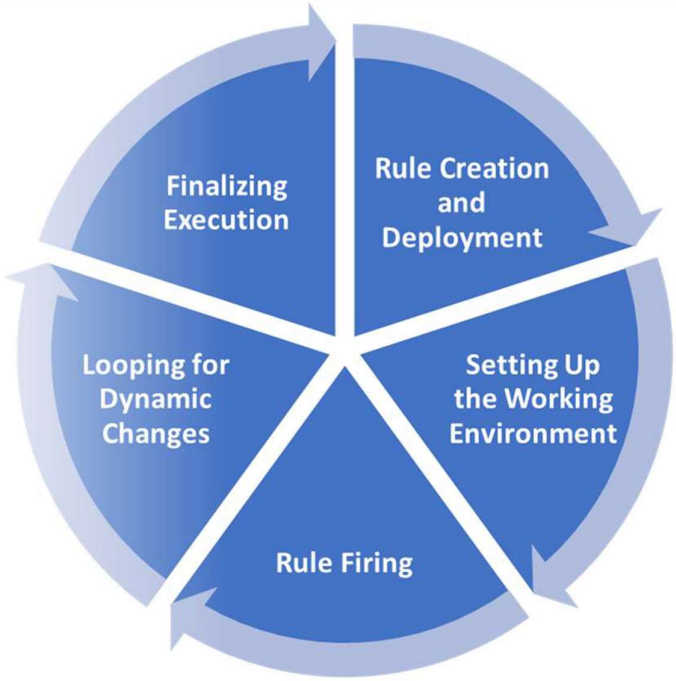
- **Re-Evaluation**: If the execution of a rule results in changes to the facts in the working memory, the rule engine may re-evaluate the rules. This is because the changes might make additional rules eligible to fire.
- **Loop Until No More Changes**: This process of evaluation, execution, and re-evaluation continues in a loop until there are no more rules to fire - either because all conditions are satisfied, or no further actions modify the facts.



The Rule Execution Lifecycle

5. Finalizing Execution

- **End of Session**: Once all applicable rules have been fired and no more changes occur in the working memory, the rule execution session ends.
- **Output/Results**: The final step involves outputting the results of the rule execution. This could be a decision, a modified data set, or any outcome defined by the rules' actions.



Common Sectors

Drools, as a versatile business rules management system, finds its applications in a variety of sectors, including:

- **Finance and Banking** – credit, fraud, compliance
- **Insurance** – claims, risk, pricing
- **E-Commerce and Retail** – pricing, promotions, inventory
- **Healthcare** – management, guidelines, compliance
- **Telecommunications** – billing, pricing, fraud detection
- **Transport and Logistics** – routing, load planning, fare
- And more...

Basic Rule Anatomy

Example Rule: "Discount for Large Orders"

Let's create a simple rule that gives a discount on large orders:

```
package com.example.rules;

rule "Discount on Large Orders"
when
    $order : Order(total > 1000) // Condition: Order total is more than 1000
then
    $order.setDiscount(10); // Action: Set a 10% discount
    System.out.println("A discount of 10% is applied for the large order.");
end
```

Basic Rule Anatomy

Let's break down a typical Drools rule and then look at a simple example with code.

Basics of Rule Anatomy in Drools

A Drools rule typically consists of the following parts:

- **Rule Name**: A unique identifier for the rule. It's like the title of a recipe in a cookbook.
- **When (LHS - Left Hand Side)**: This section contains the conditions or patterns that need to be matched for the rule to be executed.
- **Then (RHS - Right Hand Side)**: This section contains the actions to be executed when the conditions in the 'when' part are met.

```
package com.example.rules;

rule "Discount on Large Orders"
when
    $order : Order(total > 1000) // Condition:
Order total is more than 1000
then
    $order.setDiscount(10); // Action: Set a
10% discount
    System.out.println("A discount of 10% is
applied for the large order.");
end
```

Explanation

- **Package**: com.example.rules is like the namespace, grouping related rules together.
- **Rule Name**: "Discount on Large Orders" is the identifier for this rule.
- **When (LHS)**: The condition here is that the order ($order) has a total greater than 1000. The $order is a variable that refers to an instance of the Order class meeting this condition.
- **Then (RHS)**: The action is to apply a 10% discount to the order ($order.setDiscount(10)), and then print a message to the console indicating that the discount is applied.
- **End:** Denotes the end of the rule

```
package com.example.rules;

rule "Discount on Large Orders"
when
    $order : Order(total > 1000) // Conc
Order total is more than 1000
then
    $order.setDiscount(10); // Action: S
10% discount
    System.out.println("A discount of 10
applied for the large order.");
end
```

Components

- **Classes**: The Order class must be defined in your Java code, with at least two methods: **getTotal()** to retrieve the order total and **setDiscount(int)** to apply a discount to the order.
- **Rule File**: This .drl file contains the rule and is loaded into the Drools engine.
- **Execution**: When an Order object with a total greater than 1000 is inserted into the Drools session, this rule will trigger, and the corresponding actions (applying the discount and printing a message) will be executed.

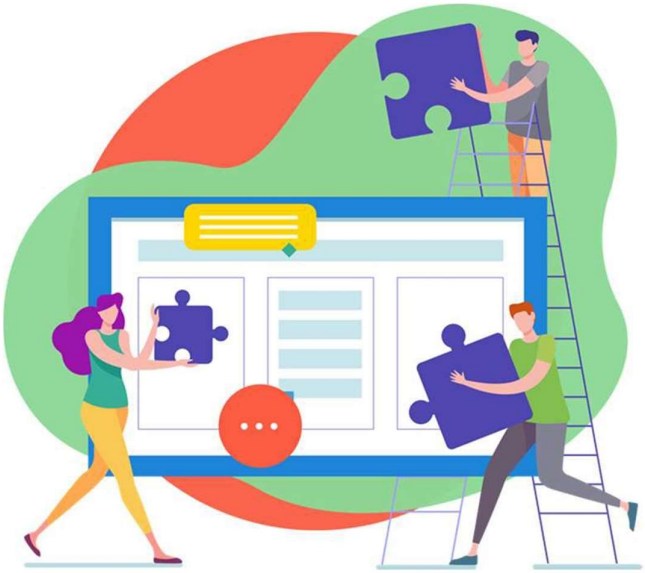
```
package com.example.rules;

rule "Discount on Large Orders"
when
    $order : Order(total > 1000) // Conc
Order total is more than 1000
then
    $order.setDiscount(10); // Action: S
10% discount
    System.out.println("A discount of 10
applied for the large order.");
end
```

Practical Exercise Overview

Introduction to Practical Exercise: Envisioning the Impact of Drools in Business Efficiency

Welcome to your first practical exercise in our journey through the world of Drools Rule Engine! Before we dive into the technical aspects of Drools in the upcoming modules, it's crucial to understand the real-world applications and the transformative impact it can have on businesses. This exercise is designed to help you bridge the gap between theoretical knowledge and practical application, fostering a deeper understanding of the significance of rule engines in today's business landscape.



Practical Exercise Overview

Why This Exercise?

In the realm of business, decision-making is a critical component. It's often complex, involves various factors, and needs to be both efficient and adaptable. This is where Drools, a powerful rule engine, comes into play. Drools automates decision-making processes, making them faster, more accurate, and consistent. However, to truly harness the power of Drools, one must first envision its application in real-life scenarios.

Practical Exercise Overview

What Will You Do?

In this exercise, you will step into the shoes of a business consultant. Your task is to identify challenges within a specific business type and then creatively develop theoretical rules that the Drools Rule Engine could implement to address these challenges. By doing so, you will explore the potential of Drools in enhancing business efficiency, improving customer experience, and streamlining operations.

Practical Exercise Overview

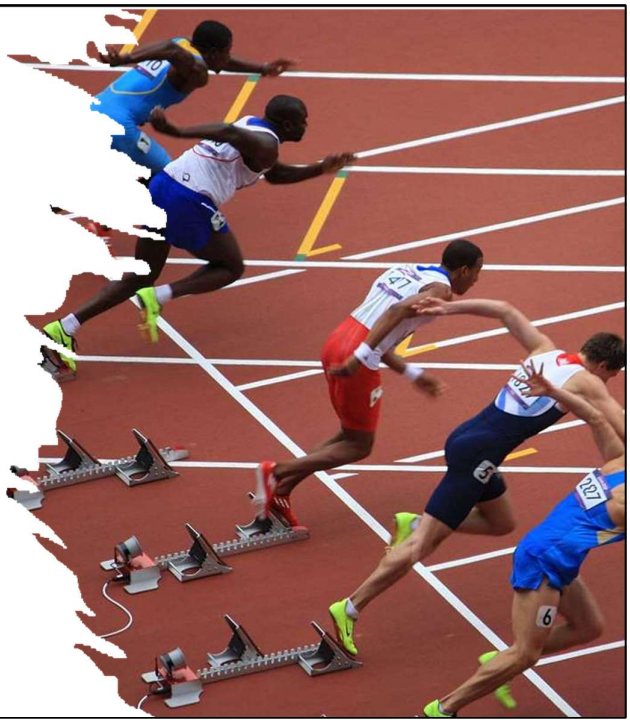
What to Expect? Through this exercise, you will:

- Develop an understanding of how rule engines can be applied in various business contexts.
- Learn to identify operational challenges that can be addressed through automated rule-based decision-making.
- Exercise creativity in formulating specific rules tailored to real-world business scenarios.
- Understand the impact of these rules in improving overall business efficiency and decision-making processes.

So, let your imagination and understanding of business processes guide you! This is your opportunity to get into the mind of a drools administrator before you start working with the tool itself.

Practical Exercise Overview

Ready? Open and Complete Lab
Exercise 01 – Module 01 Practical Task



Quiz Time!

Module Summary

- **Understanding Rule Engines:** We explored what rule engines are and how they function.
- **Historical Context and Evolution of Rule Engines:** The history of rule engines highlighted their evolution from simple decision-making tools to sophisticated automation systems.
- **Procedural vs. Declarative Programming:** We discussed how procedural programming focuses on the 'how', and how declarative programming emphasizes the 'what'.
- **Case Studies: Appropriate Use Cases for Rule Engines:** We examined various scenarios where rule engines shine.
- **An Introduction to the Drools Platform:** We introduced Drools, discussing its open-source nature and its connection to Red Hat.
- **The Lifecycle of Rule Execution:** We delved into the inner workings of a rule engine, from creation and deployment to execution and output.
- **Basics of Rule Anatomy in Drools:** We focused on the structure of rules in Drools, covering the key components such as the rule name, LHS (conditions), and RHS (actions).
- **Practical Exercise:** We put on our thinking caps to imagine how to use drools in a practical business scenario
- **Interactive Quiz: Assessing Understanding:** We engaged in an interactive quiz, testing our understanding of rule engines and the Drools platform.

DROOLS 8

MODULE 02:
SYSTEM SETUP

PREPARING THE ENVIRONMENT

Presented by John Paul Franke

Course Overview

Welcome to Module 02: Setting Up Your Drools Environment!

In this module, we're embarking on a practical journey, setting up the foundation for all our Drools projects. We'll be organizing and configuring our software tools and environment for Drools.

This module is designed to guide you through the process of installing and configuring all the necessary components to get your Drools environment up and running. Whether you are an experienced developer or new to the world of rule engines, this module will provide step-by-step instructions to ensure a smooth setup.

Learning Objectives

What Will You Learn?

- The prerequisites needed for a successful Drools installation.
- How to set up your system for successful running of Drools
- A detailed guide on installing Drools dependencies.
- Understanding Maven and how it integrates with Drools.
- Configuring your Integrated Development Environment (IDE) for Drools, with a focus on IntelliJ IDEA and Visual Studio Code (VSCode).
- How to ensure your environment is correctly set up and ready for developing Drools applications.

By the end of this module, you'll have a fully functional Drools development environment, ready to create and run rule-based applications. This setup is crucial as it lays the groundwork for all the exciting Drools features and projects we'll explore in the subsequent modules.

Let's get our tools ready and set the stage!

Drools Prerequisites

Before diving into Drools, it's important to ensure that you have the necessary prerequisites in place.

Here are the key prerequisites for working with Drools:

- **Java Development Kit (JDK):**
  - Drools is a Java-based framework, so you need to have Java installed on your computer.
  - The recommended version is JDK 17, which you can download and install from providers like Oracle or adopt OpenJDK versions such as Amazon Corretto.
  - Ensure that the **JAVA_HOME** environment variable is correctly set to the JDK installation path.

- **Integrated Development Environment (IDE):**
  - An IDE makes it easier to write, debug, and manage your Java and Drools code.
  - Popular choices include IntelliJ IDEA and Visual Studio Code (VSCode). Both have excellent support for Java and Drools.
  - These IDEs offer features like code completion, syntax highlighting, and debugging tools, which are extremely helpful when working with Drools.

- **Apache Maven:**
  - Maven is a build automation tool used primarily for Java projects.
  - It manages project dependencies and simplifies the build process for Drools applications.
  - Maven ensures that all necessary libraries for Drools are downloaded and included in your project.

Intro: Lab 02 – Java Installation

Welcome to Lab Exercise 02: Java Installation

In this next lab exercise, we'll embark on the initial step of our Drools journey - setting up Java on your system. Java is the backbone of Drools, serving as the foundational platform upon which all Drools applications are built. Think of it like laying the first brick or foundation stone for a building; without this, we can't construct our structure of rules and logic.

Intro: Lab 02 – Java Installation

Prerequisites:

- A computer running Windows OS (Windows 7 or later).
- An active internet connection for downloading the JDK.
- Basic knowledge of navigating Windows and using a web browser.

Learning Objectives:

- How Download and install a JDK suitable for Drools.
- Configure the `JAVA_HOME` environment variable, a crucial step for Java and Drools to function correctly on your system.
- How to verify your Java installation to ensure that it is ready for use with Drools.

Intro: Lab 02 – Java Installation

Objective:

Our goal in this lab is to guide you through installing a compatible Java Development Kit (JDK), a critical component needed to run Drools. This exercise is tailored to ensure you have a properly configured Java environment, setting the stage for seamless future development with Drools.

Why Is This Important?

Without Java, Drools cannot run. Setting up Java correctly is not only vital for Drools but also beneficial for your broader journey in the world of Java-based applications. This lab will provide you with the necessary skills and knowledge to prepare your system for running Java applications, including those built with Drools.

Intro: Lab 02 – Java Installation

Time to open and complete Lab 02 – Java Installation

Intro: Lab 03 – Maven Installation

Welcome to Lab Exercise 03: Maven Installation

After setting up Java in the first lab, we now proceed to another crucial step in our Drools journey - installing Apache Maven. Maven plays a pivotal role in managing and streamlining the build process for Java projects, including those involving Drools.

Objective: In this lab, we will guide you through the installation and verification of Apache Maven on your system. Maven is a powerful tool for project management and build automation, and it is especially important for handling dependencies and build processes in Drools.

Intro: Lab 03 – Maven Installation

Prerequisites:

- Successful completion of Lab Exercise 01 (Java Installation).
- A computer with a Windows operating system (Windows 7 or later).
- Internet access to download the Maven package.
- Basic understanding of navigating through Windows directories and using the command line interface.

Learning Objectives:

- How to download and install Apache Maven, an essential tool for Java and Drools development.
- The process of adding Maven to your system's PATH environment variable, making it accessible from the command line.
- How to verify that Maven has been correctly installed and configured on your system.

Intro: Lab 03 – Maven Installation

Why Maven?

Maven simplifies the management of Java projects by automating tasks like building, reporting, and documentation. For Drools, Maven helps in efficiently handling project dependencies, ensuring that all necessary libraries and tools are in place for your Drools projects.

Outcomes:

Upon completing this lab, you'll have a fully functional Maven installation on your system. This setup will enable you to manage Drools projects and dependencies with ease, setting a strong foundation for more advanced work with Drools in the upcoming modules.

Intro: Lab 03 – Maven Installation

Let's do it! Open and complete Lab 03 – Maven Installation

Intro: Lab 04 – IntelliJ IDEA Installation

Welcome to Lab Exercise 04: Installing IntelliJ IDEA

Having successfully installed Java and Maven, you are now ready to set up your Integrated Development Environment (IDE). In this lab, we'll focus on installing IntelliJ IDEA, a popular IDE among Java developers, and a great tool for working with Drools. IntelliJ IDEA provides a powerful and user-friendly interface for developing Java and Drools applications, making your coding experience more efficient and enjoyable.

Intro: Lab 04 – IntelliJ IDEA Installation

Prerequisites:

- Completion of Lab Exercises 01 (Java Installation) and 02 (Maven Installation).
- A Windows-based computer (Windows 7 or later).
- An internet connection to download IntelliJ IDEA.
- Basic familiarity with operating a Windows computer and navigating the internet.

What You Will Achieve:

- Download and install either the Community or Ultimate version of IntelliJ IDEA.
- Familiarize yourself with the IntelliJ IDEA interface and its various features.
- Set up a new Java project in IntelliJ IDEA to verify the installation.

Intro: Lab 04 – IntelliJ IDEA Installation

Why IntelliJ IDEA?

IntelliJ IDEA is renowned for its robust set of features, including intelligent coding assistance, ergonomic design, and powerful tools for Java developers. It's particularly advantageous for Drools development due to its advanced code navigation and debugging capabilities.

Outcomes:

By the end of this lab, you will have a fully installed and configured IntelliJ IDEA environment on your Windows system. This setup will serve as your central platform for developing Java and Drools applications throughout this course and beyond.

Intro: Lab 04 – IntelliJ IDEA Installation

Hands on deck! Open and complete Lab Exercise 04 – IntelliJ IDEA Installation

Quiz Time!

Module Recap

Here's a recap of what we've covered in this module:

- **Introduction to Drools Environment Setup:** We began with an overview of what to expect in this module, emphasizing the importance of a properly configured development environment for Drools.
- **Prerequisites for Drools:** You learned about the essential prerequisites for working with Drools. This included the Java Development Kit (JDK), an Integrated Development Environment (IDE), Apache Maven
- **How to set up your system for successful running of Drools:** We covered how to get your system ready to run Java, your JDK, and Maven
- **Hands-on Lab Exercises:** The module included several lab exercises where you applied what you learned:
  - **Lab Exercise 01:** Focused on installing and verifying Java on your system.
  - **Lab Exercise 02:** Covered the installation and verification of Apache Maven.
  - **Lab Exercise 03:** Guided you through installing and setting up IntelliJ IDEA.

These exercises were designed to give you practical experience and ensure that you have a fully functional development environment for Drools.

DROOLS 8

MODULE 03:
DROOLS SETUP

BUILDING A DROOLS PROJECT

Presented by John Paul Franke

Course Overview

Welcome to Module 03 of our Drools Rule Engine course!

Having successfully navigated the foundational concepts of rule engines and the essential setup of your Drools environment, you are now ready to venture into the practical aspects of building, deploying, utilizing, and running Drools applications.

In this module, we will be delving into the heart of Drools, exploring how to construct, deploy, and utilize Drools within your applications. Like a chef who has organized their kitchen and is now ready to start cooking complex dishes. You've set up your 'kitchen'; now it's time to start 'cooking' with Drools!

Learning Objectives

What Will You Learn?

- **Constructing a Drools Maven Project:** You'll learn how to assemble the components of a Drools project using Maven.
- **Understanding KIE Sessions:** Discover the importance of KIE sessions in Drools, and understand stateful vs stateless sessions
- **Rule Syntax and Creation:** You'll delve deeper into writing rules in Drools, learning the syntax and structure.
- **Developing and Testing Rules:** You'll learn to develop and test rules to ensure they perform as expected in real-world scenarios.
- **Deploying and Running Drools Applications:** Finally, you'll learn how to deploy and run your Drools applications in different IDE's!

Learning Objectives

Why is This Important?

Understanding the construction and execution of Drools applications is crucial. It enables you to automate complex decision-making processes in your applications, enhancing efficiency and accuracy. This module is your stepping-stone to becoming proficient in utilizing Drools in real-world scenarios.

Understanding KIE Sessions

What is a KIE?

KIE, which stands for "Knowledge Is Everything", is a central concept in the Drools ecosystem. It represents the comprehensive suite of tools and services for knowledge management provided by the Drools platform, including a repository of all the knowledge definitions that you have in your Drools project. This includes rules, processes, functions, and types. In essence, KIE is the umbrella under which various Drools components and services are grouped.

Understanding KIE Sessions

What is a KIE Session?

A KIE Session in Drools is a core concept that serves as a bridge between your data and the rules you've defined in Drools. In simple terms, a KIE Session in Drools is like a workspace where your rules and data come together to make decisions.

Understanding KIE Sessions

Here's a step-by-step breakdown of how a typical KIE Session runs in Drools:

Step 1:

Create a KIE Session: The first step is to define a KIE Session parameters in a `kmoodule.xml` file. You then launch the session in KIE Container, which contains all the knowledge bases and sessions defined in your Drools project.

Understanding KIE Sessions

Step 2:

Insert Data (Facts): Once the session is created, you insert data into it. These pieces of data are known as "facts." Facts are the objects that the rules will act upon. You can insert as many facts as needed, and they can be of any complexity, from simple data types to complex Java objects.

Understanding KIE Sessions

Step 3:

Fire Rules: After inserting the facts, you instruct the session to fire the rules. This is when the Drools engine starts processing the facts against the rules defined in your rule files (DRL files). The engine checks each rule to see if its conditions (defined in the 'when' part of the rule) are met by the current facts in the session. If the conditions are satisfied, the actions (defined in the 'then' part of the rule) are executed.

Understanding KIE Sessions

Step 4:

Rules Execution: The rules are executed in an order based on their priority (defined using 'salience') and the order in which facts match the rules. In a stateful session, the engine keeps track of which rules have been fired and the state of the facts. This allows for rules to react to changes made by other rules.

Understanding KIE Sessions

Step 5:

Modify or Update Facts: During the execution of the session, your rules might modify the facts or insert new facts. In a stateful session, these changes are remembered. If a fact is modified, the engine re-evaluates the rules to see if the changes lead to new rules being fired.

Understanding KIE Sessions

Step 6:

Repeat Rule Evaluation (if necessary): The process of firing rules and modifying facts can happen multiple times in a loop. The engine continues to re-evaluate and fire rules as long as facts are being changed and rules are being triggered.

Understanding KIE Sessions

Step 7:

Halt the Session (if needed): In some scenarios, you might want to programmatically stop the session from processing further. This can be done by calling a method to halt the session within the actions of a rule.

Understanding KIE Sessions

Step 8:

Dispose of the Session: Once all the rules have been processed and no more rules are eligible to fire, it's a good practice to dispose of the session. Disposing of the session releases the resources associated with it and mitigates the risk of memory leaks.

Understanding KIE Sessions

Step 9:

Extract Results: After the session is completed, you can extract the results of the rule execution, which could be in the form of modified facts, new facts, or any other outcomes defined by your rules.

Intro: Lab 05 – Drools Project Setup

Welcome to Lab Exercise 05: Drools Project Setup!

Now that you have Java, Maven, and IntelliJ IDEA installed, it's time to venture into the core of this course - setting up your first Drools project. This lab is where things really get started, as you'll be taking your first steps in Drools development.

Objective: In this lab, you will learn how to create a basic Drools project using IntelliJ IDEA. We'll go through the process of configuring a Drools project, creating a simple rule, and verifying the setup. This exercise is your first hands-on introduction to working with Drools rules and the Drools engine.

Intro: Lab 05 – Drools Project Setup

Prerequisites:

- A Windows-based computer (Windows 7 or later).
- An internet connection to download IntelliJ IDEA.
- Corresponding files `SetupCodes.docx` and `dependencies.docx`
- Basic familiarity with operating a Windows computer and navigating the internet.

What You Will Learn:

- How to create a new project in IntelliJ IDEA with Maven configuration for Drools.
- Setting up the `pom.xml` file with necessary Drools dependencies.
- Creating the directory structure and files required for a Drools project, including a basic Drools rule file.
- Understanding the function and configuration of the `kmodule.xml` file.
- Writing and running a simple Java class to execute a Drools session and verify the setup.

Intro: Lab 05 – Drools Project Setup

Why This Exercise? Setting up a Drools project correctly is crucial for all future work in this course. This exercise will familiarize you with the basic structure of a Drools project, how Drools rules are defined and executed, and how Java interacts with Drools.

Outcomes: By completing this lab, you'll have a foundational Drools project ready on your system. You will gain the confidence and skills to start exploring more complex aspects of Drools in the subsequent modules.

Intro: Lab 05 – Drools Project Setup

Roll up those sleeves! Open and complete Lab Exercise 05 – Drools Project Setup

Intro: Lab 06 – Drools Setup with VSCode

Welcome to Lab Exercise 06: Drools Setup with Visual Studio Code (VSCode)

After exploring IntelliJ IDEA, we turn our attention to another popular Integrated Development Environment (IDE) - Visual Studio Code (VSCode). In this lab, you'll learn how to set up a Drools project using VSCode, a lightweight, powerful, and highly customizable IDE. This exercise will demonstrate that Drools development can be effectively managed in various development environments.

Objective: This lab guides you through the process of setting up a Drools project in VSCode. You'll configure your project using Maven, install necessary extensions, and test a basic Drools rule to ensure everything is working correctly.

Intro: Lab 06 – Drools Setup with VSCode

Prerequisites:

- Completion of Lab Exercise 01 (Java Installation) and Lab Exercise 02 (Maven Installation).
- A Windows computer with internet access
- Corresponding files `SetupCodes.docx` and `dependencies.docx`
- VSCode installed on your system.

What You Will Achieve:

- Installing essential VSCode extensions for Drools and Java development.
- Creating and configuring a Java project in VSCode with Maven support.
- Setting up the necessary project structure for a Drools application.
- Creating a basic Drools rule file and configuring the `kmodule.xml` file.
- Running and testing the Drools project within VSCode.

Intro: Lab 06 – Drools Setup with VSCode

Why VSCode?

VSCode offers a more lightweight and flexible environment compared to traditional Java IDEs like IntelliJ IDEA. It's preferred by developers who enjoy a faster setup and customizable experience, while still providing powerful features for Java and Drools development.

Outcomes:

Upon completing this lab, you'll have a Drools project set up in VSCode, ready for further exploration and development. This setup paves the way for working on more complex Drools applications and provides a solid understanding of how Drools integrates with different development tools.

Intro: Lab 06 – Drools Setup with VSCode

One more to go! Open and Complete Lab Exercise 06 – Drools Setup with VSCode

Quiz Time!

Module Recap

Here's a recap of the key highlights of module 03:

- **Constructing a Drools Maven Project in IntelliJ IDEA and VSCode**: We explored the steps to create a Drools project using Maven in both the IntelliJ IDEA and Visual Studio Code (VSCode) IDE's.
- **Understanding KIE Sessions**: We learned the inner workings and how to define, create, and run a basic KIE session
- **Fundamentals of Rule Syntax in Drools**: We learned the correct syntax for creating a basic rule in DRL
- **Developing and Testing Rules**: We successfully fired our first rule, verifying our system and rule setup
- **Interactive Quizzes and Practical Exercises**: To reinforce learning, the module included interactive quizzes and practical exercises.

We hope you've enjoyed this course so far! Join us next week to delve deeper into the creation and management of Drools Rules!