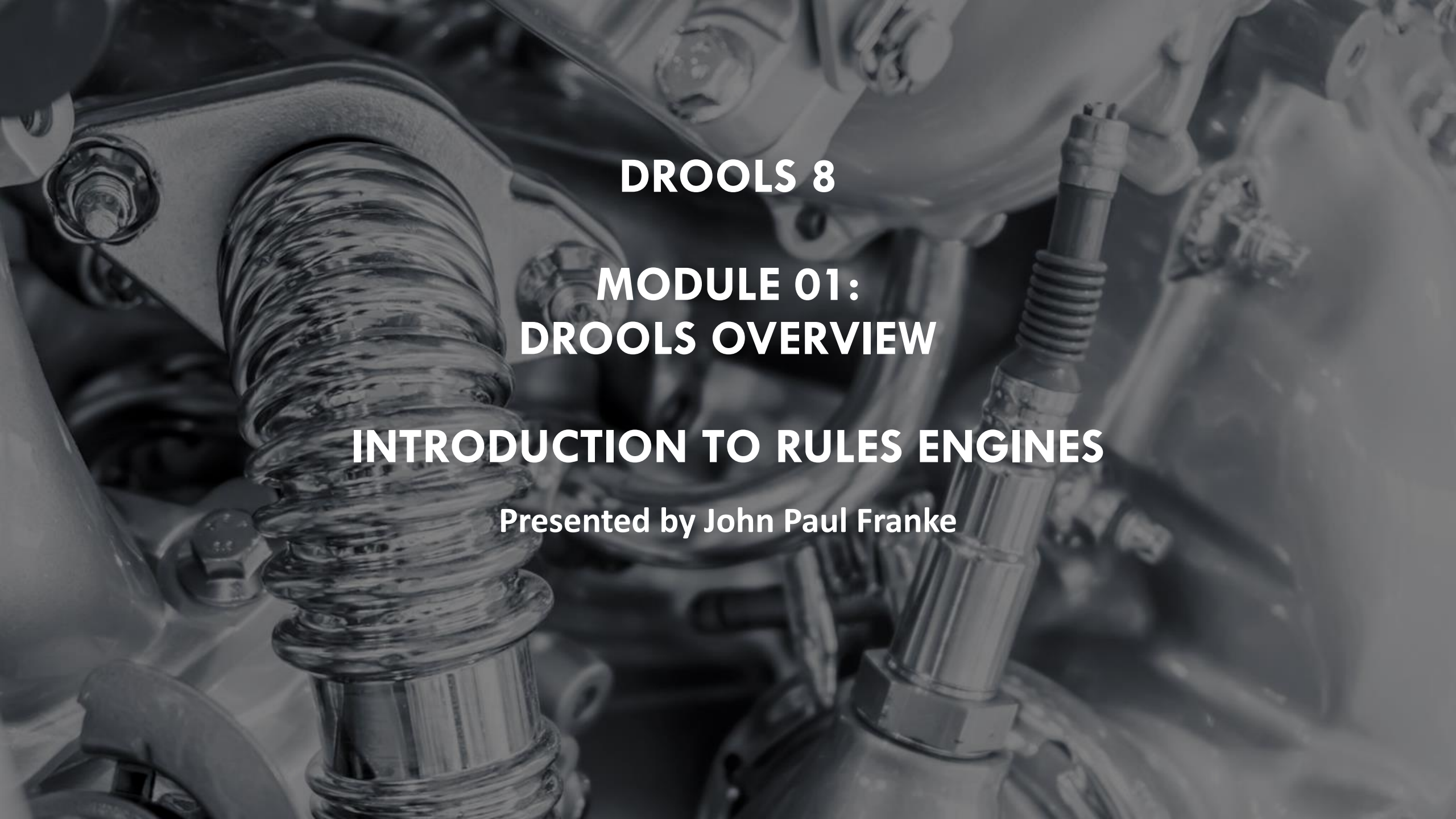


DROOLS 8

MODULE 01: DROOLS OVERVIEW

INTRODUCTION TO RULES ENGINES

Presented by John Paul Franke

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

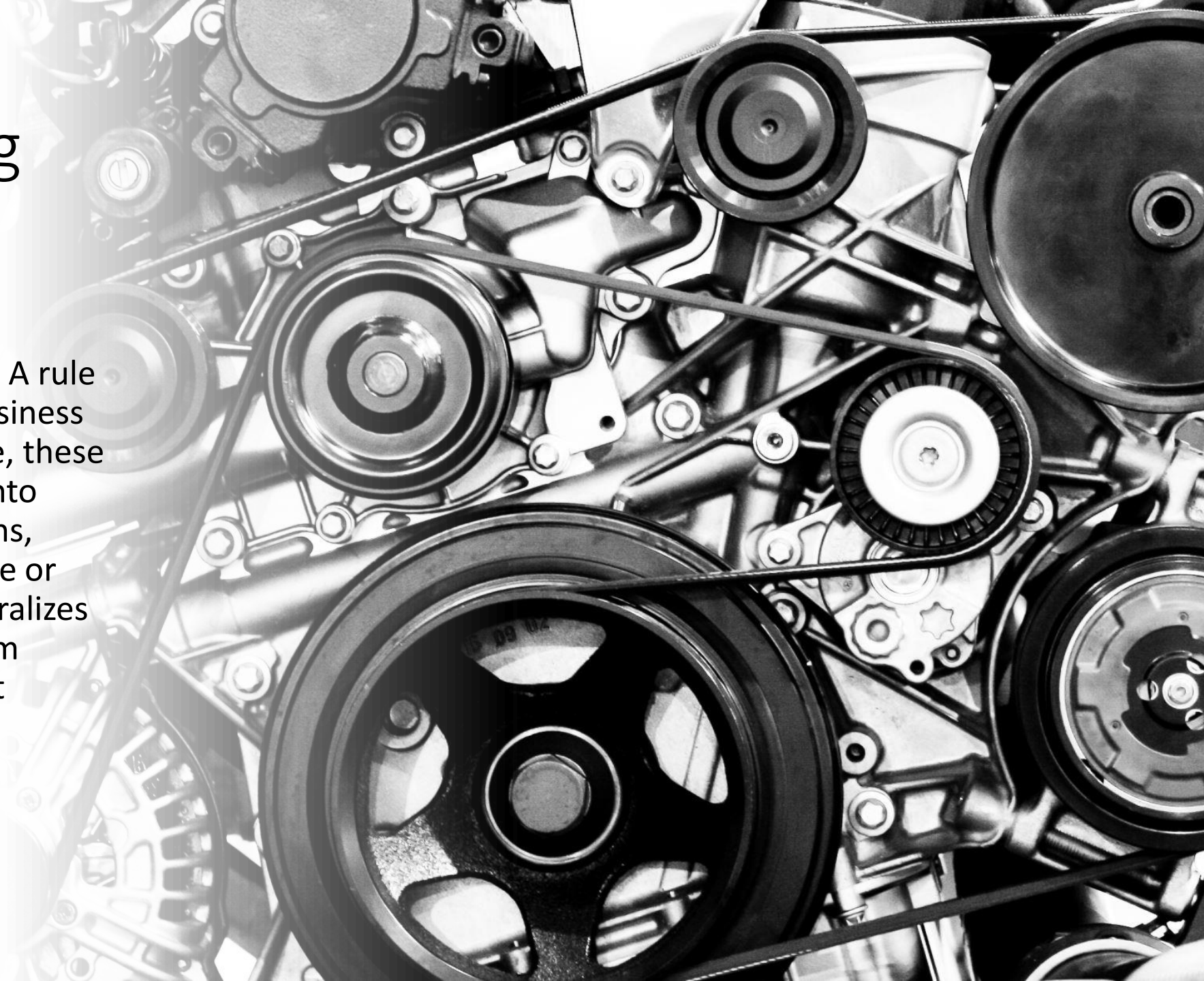
Rule engines can simplify complex decision-making. They keep the rules separate from the rest of the program, making it easier to update and manage them, just like how a chef can easily update the recipe book without changing the entire kitchen setup. This makes businesses more agile, able to adapt quickly to new information or changing conditions.

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.

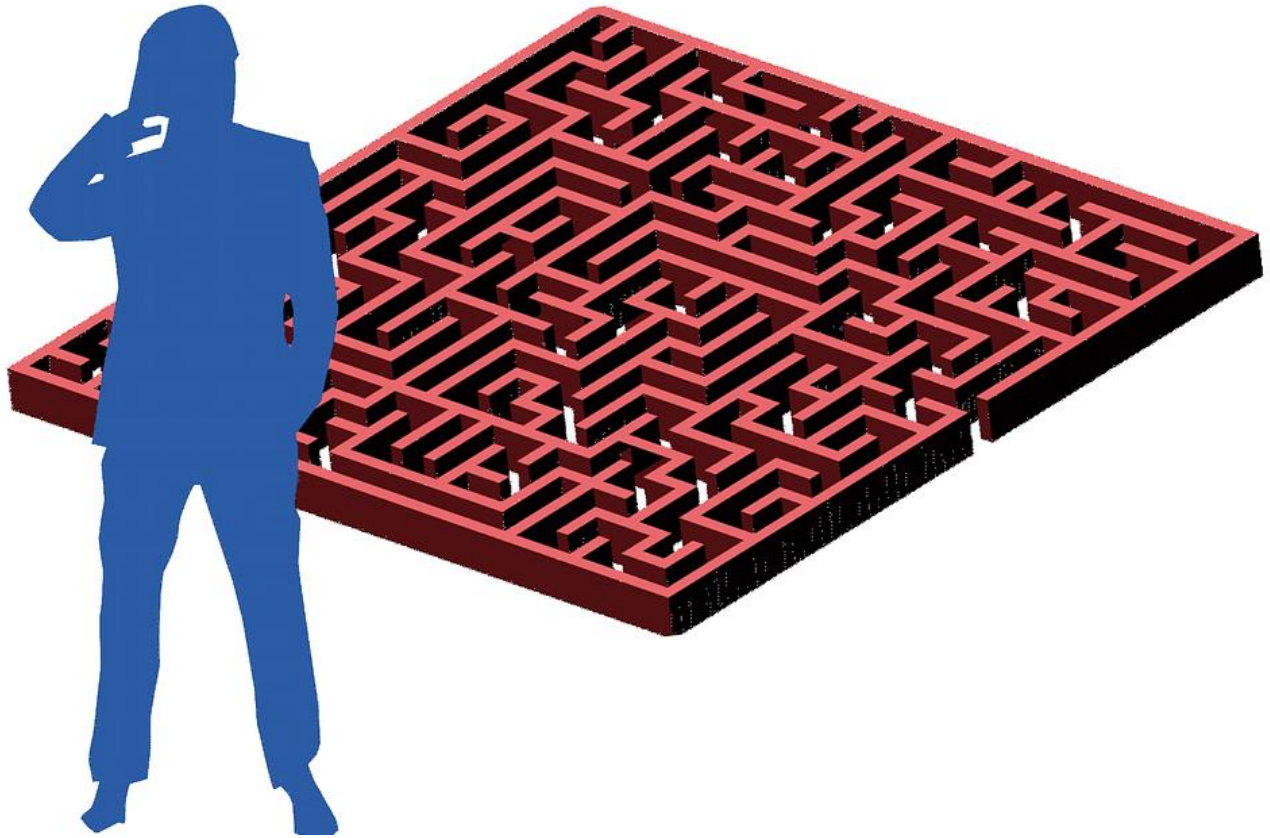
Understanding Rule Engines

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.

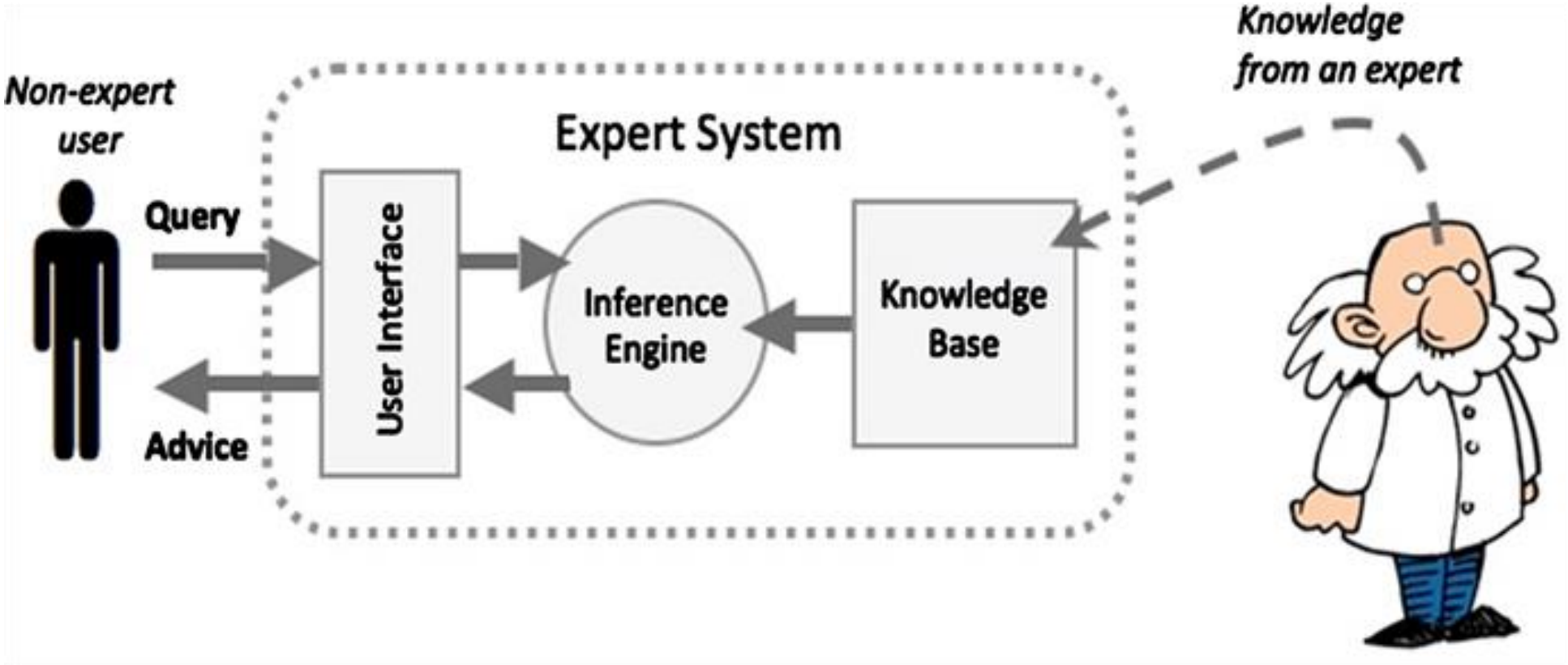


History & Evolution of R.E.'s

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

When to Use Rule Engines

Rule Engines can greatly enhance automation and efficiency is a business, but they may not always be the right solution. Let's examine situations where rule engines may be most useful...

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

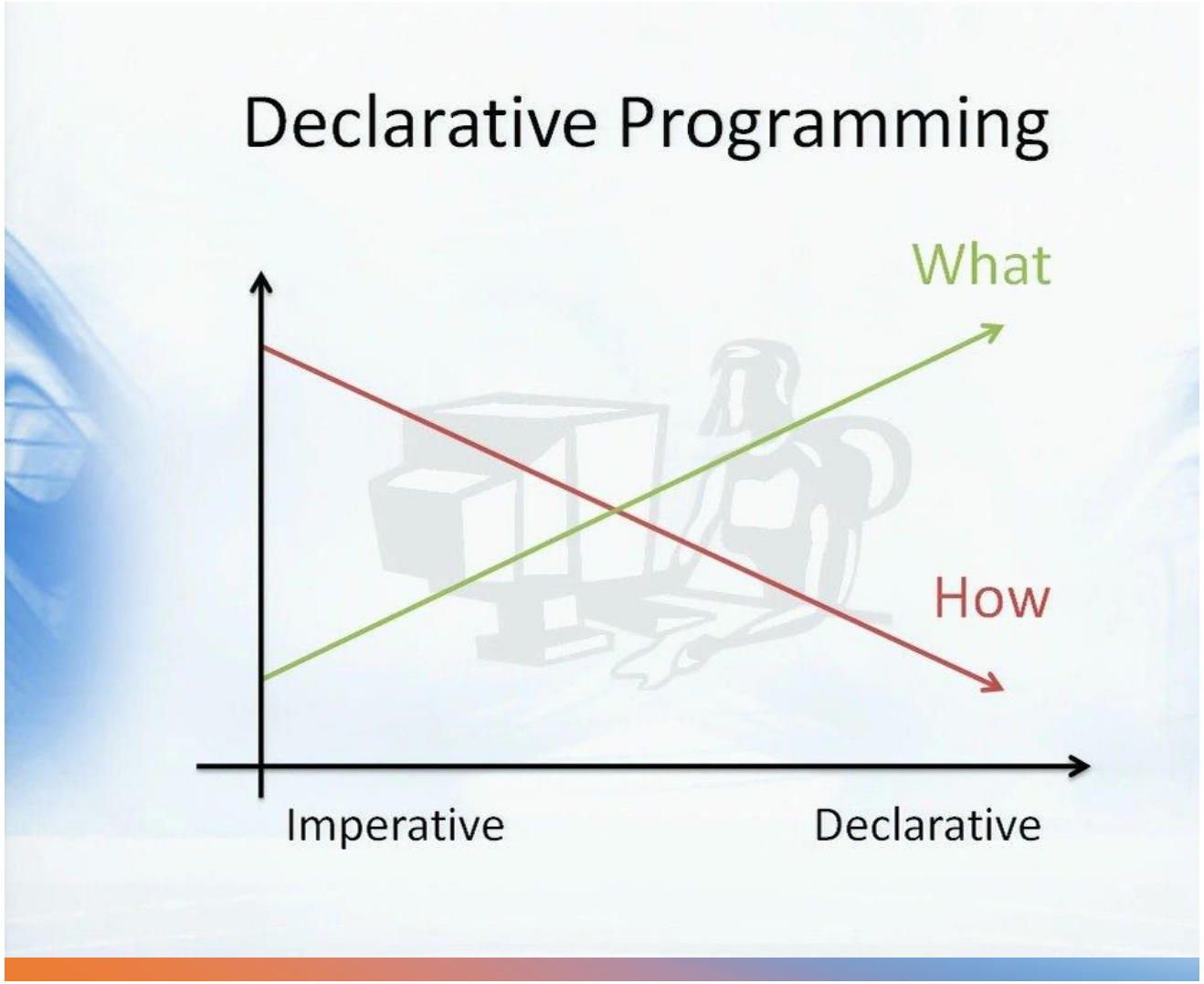
Declarative Programming:

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:

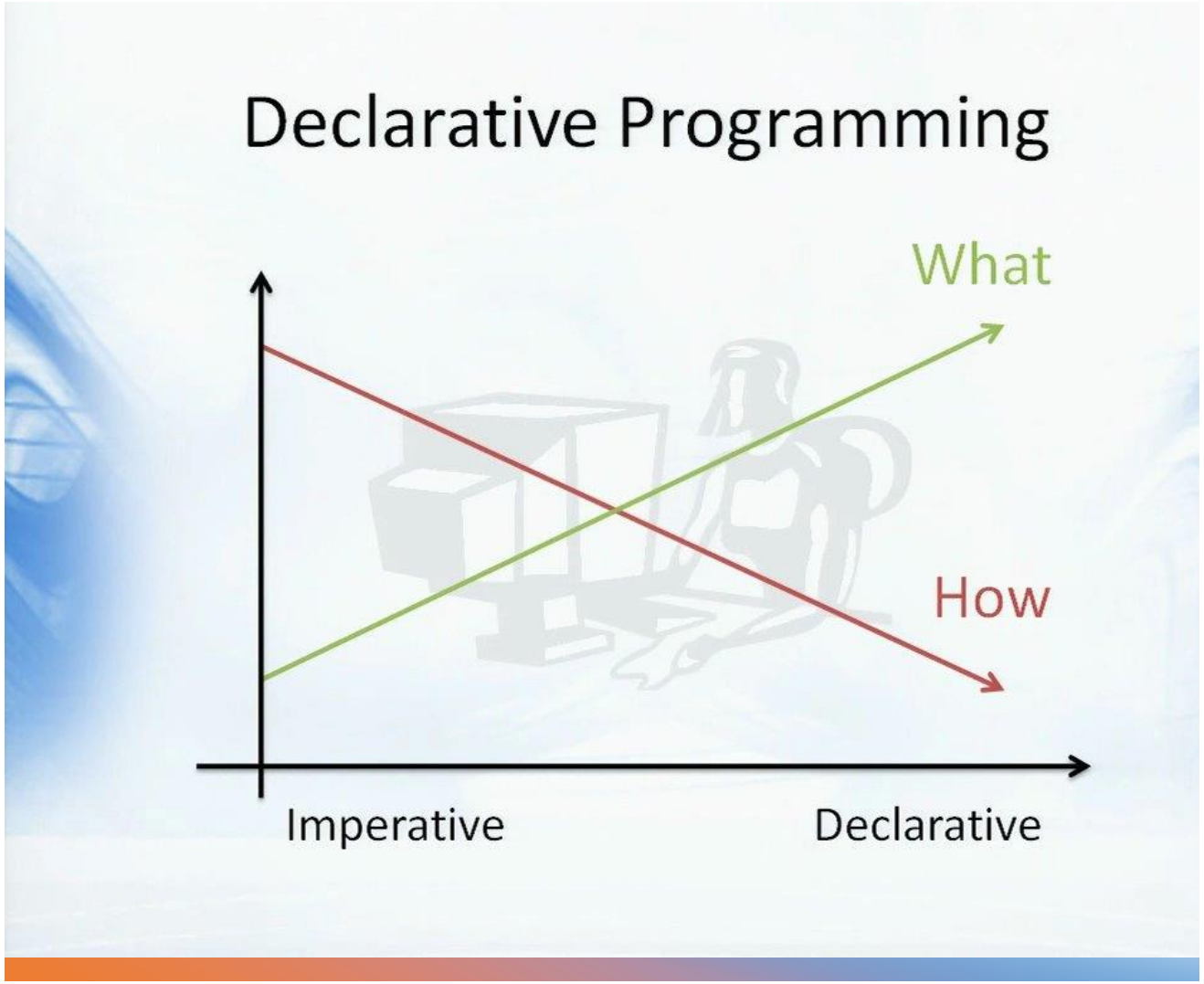
- **Control**: In procedural programming, you control the process, like following a recipe. In declarative programming, you control the outcome, like ordering a sandwich.



Procedural vs Declarative Programming

Key Differences:

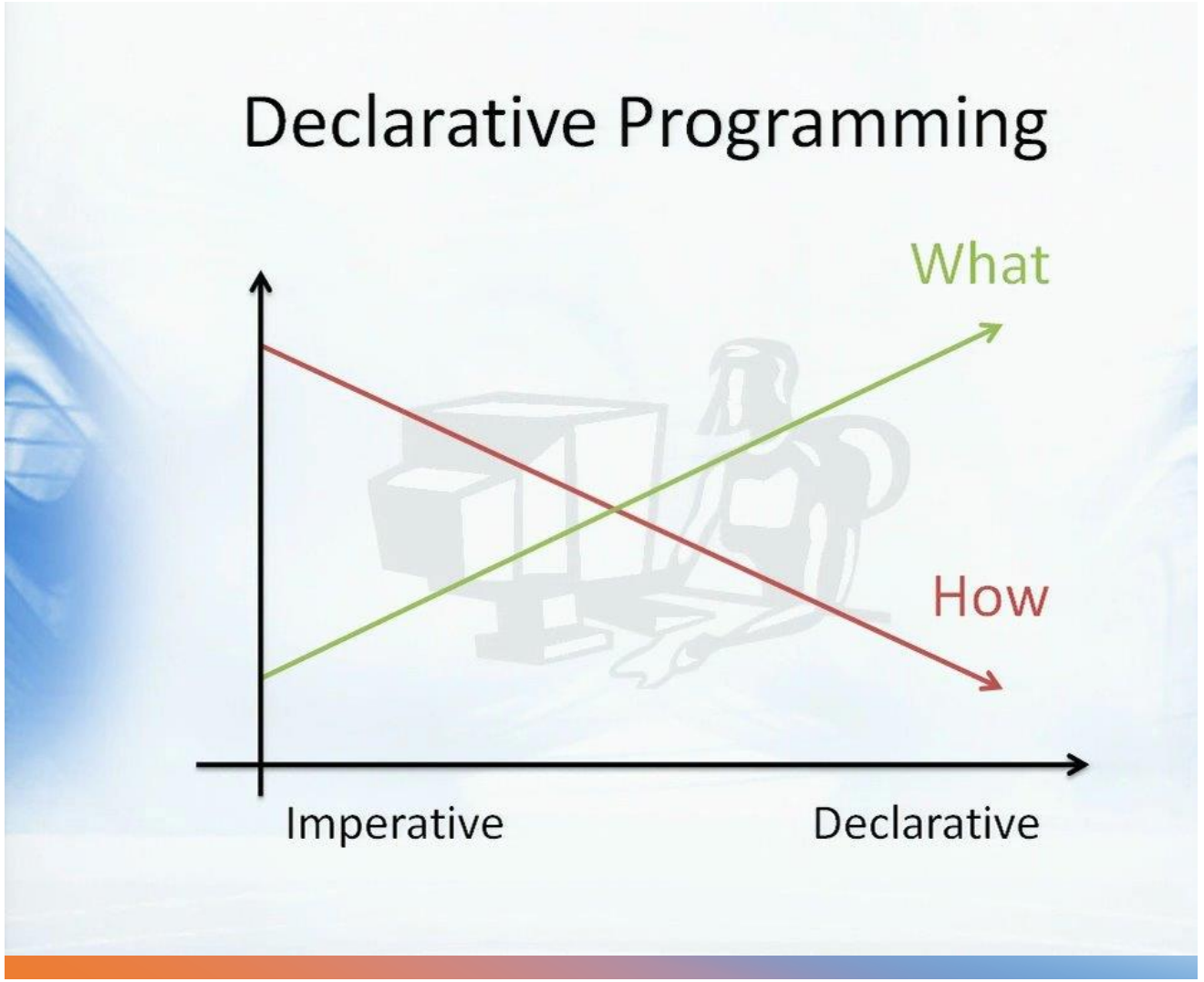
- **How vs. What**: Procedural focuses on 'how' to do things (the process), while declarative focuses on 'what' you want to achieve (the outcome).



Procedural vs Declarative Programming

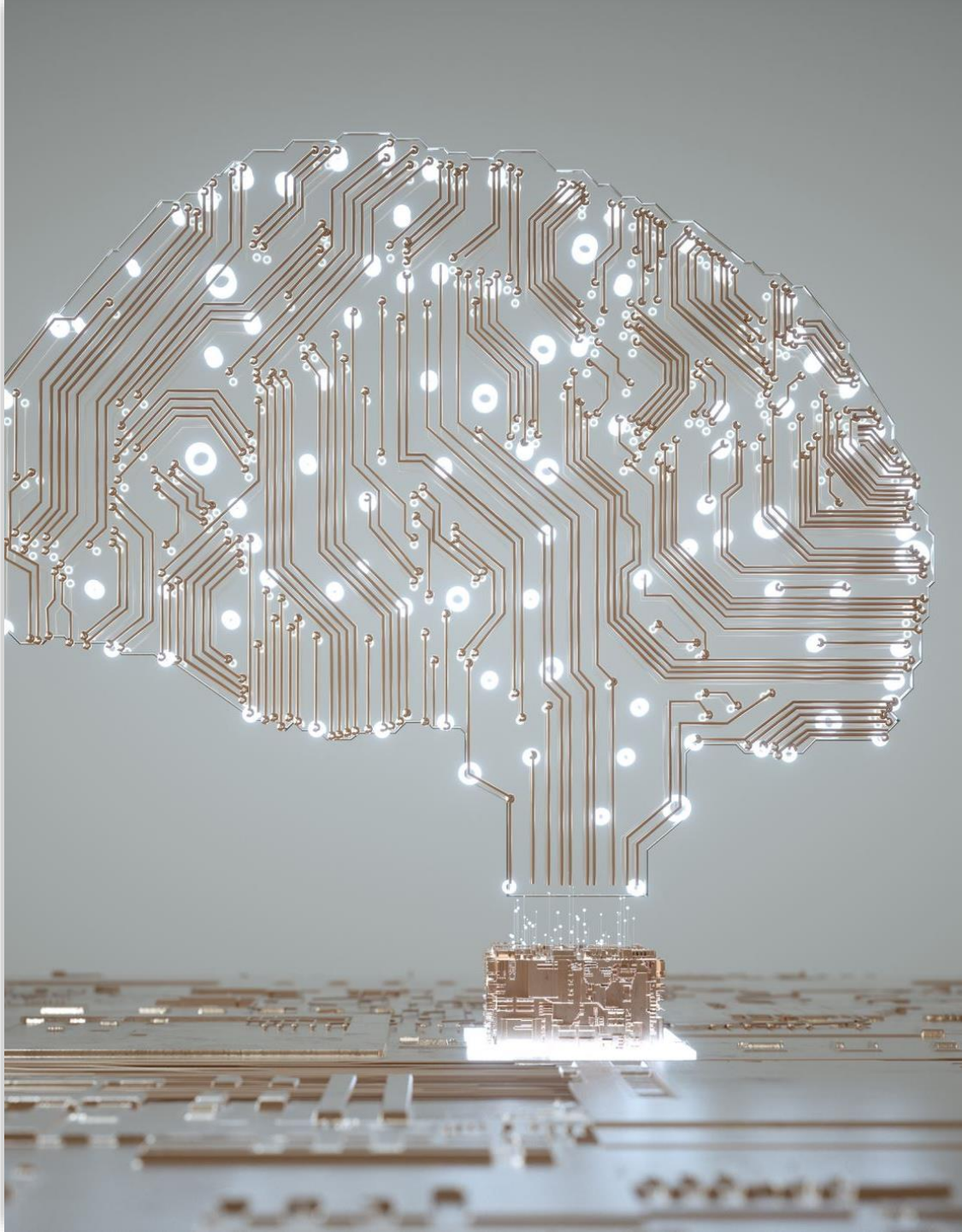
Key Differences:

- **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

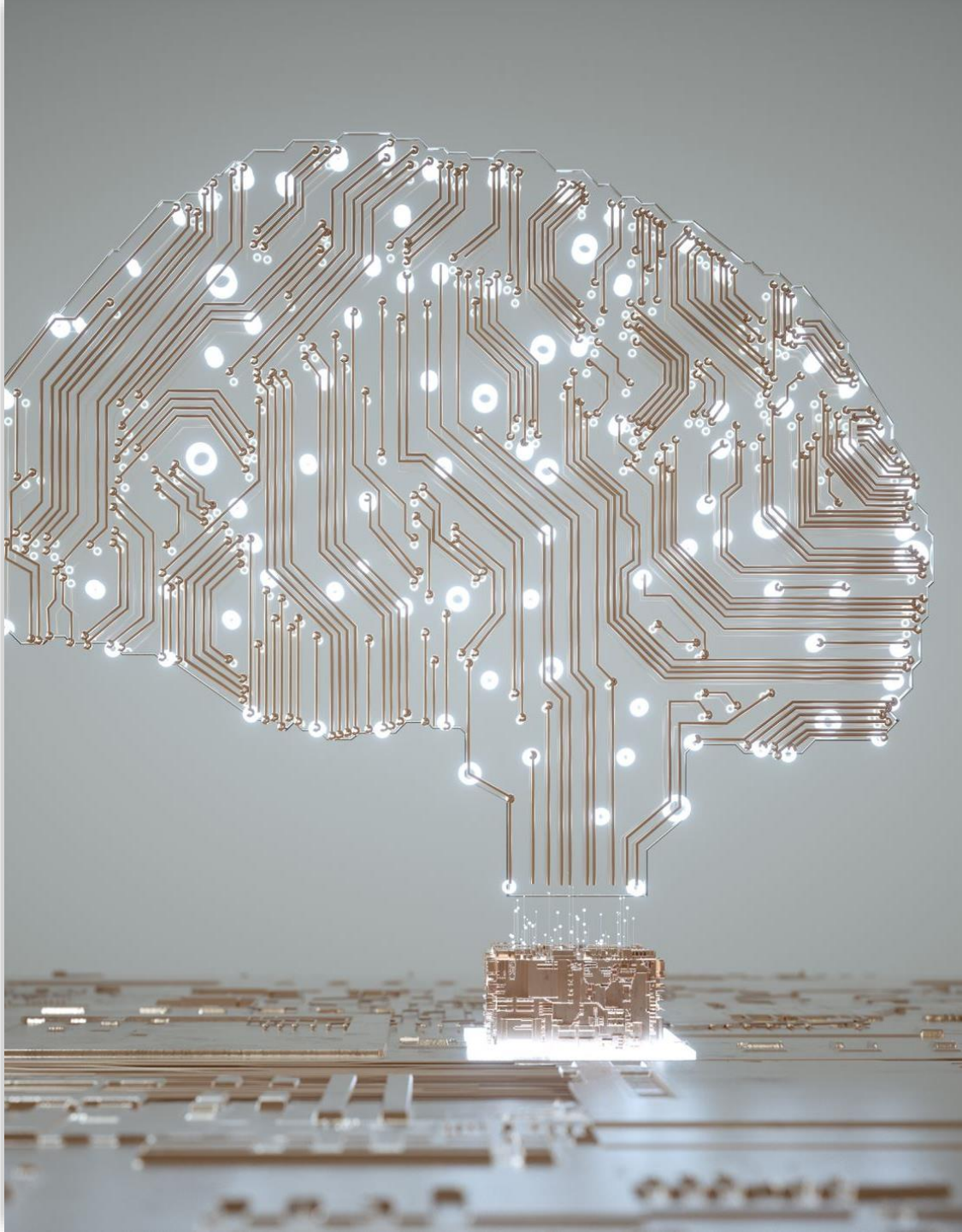
Rule engines are often considered an early form of Artificial Intelligence (AI) because of their ability to mimic certain aspects of human decision-making using predefined logic and rules.



R.E's as early AI

Mimicking Human Decision-Making

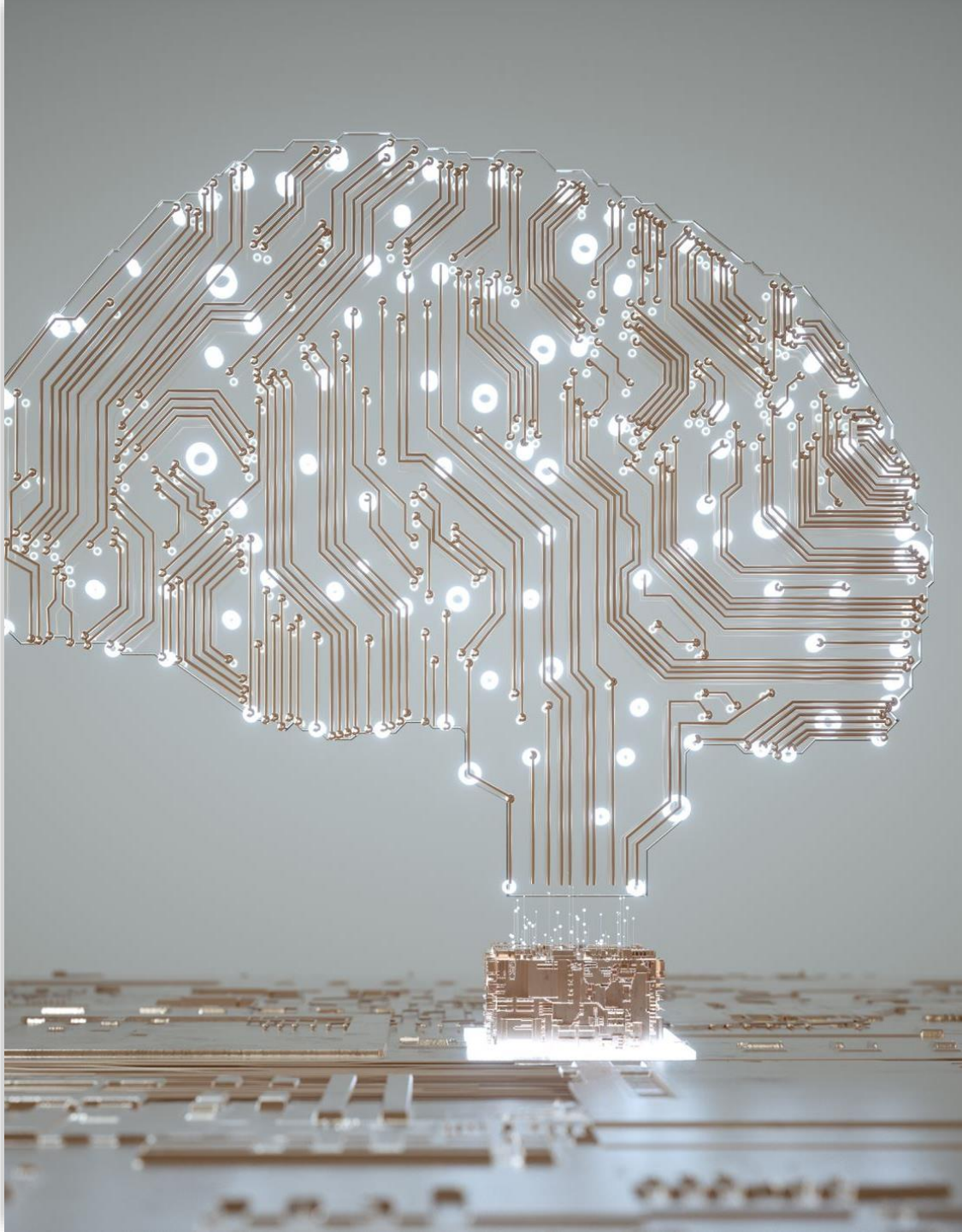
- **Expert Systems**: Expert systems were among the first successful forms of AI. These systems sought to emulate the decision-making ability of human experts. By encoding expert knowledge in the form of rules, these systems could make informed decisions, similar to how a human expert would.
- **Logic 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

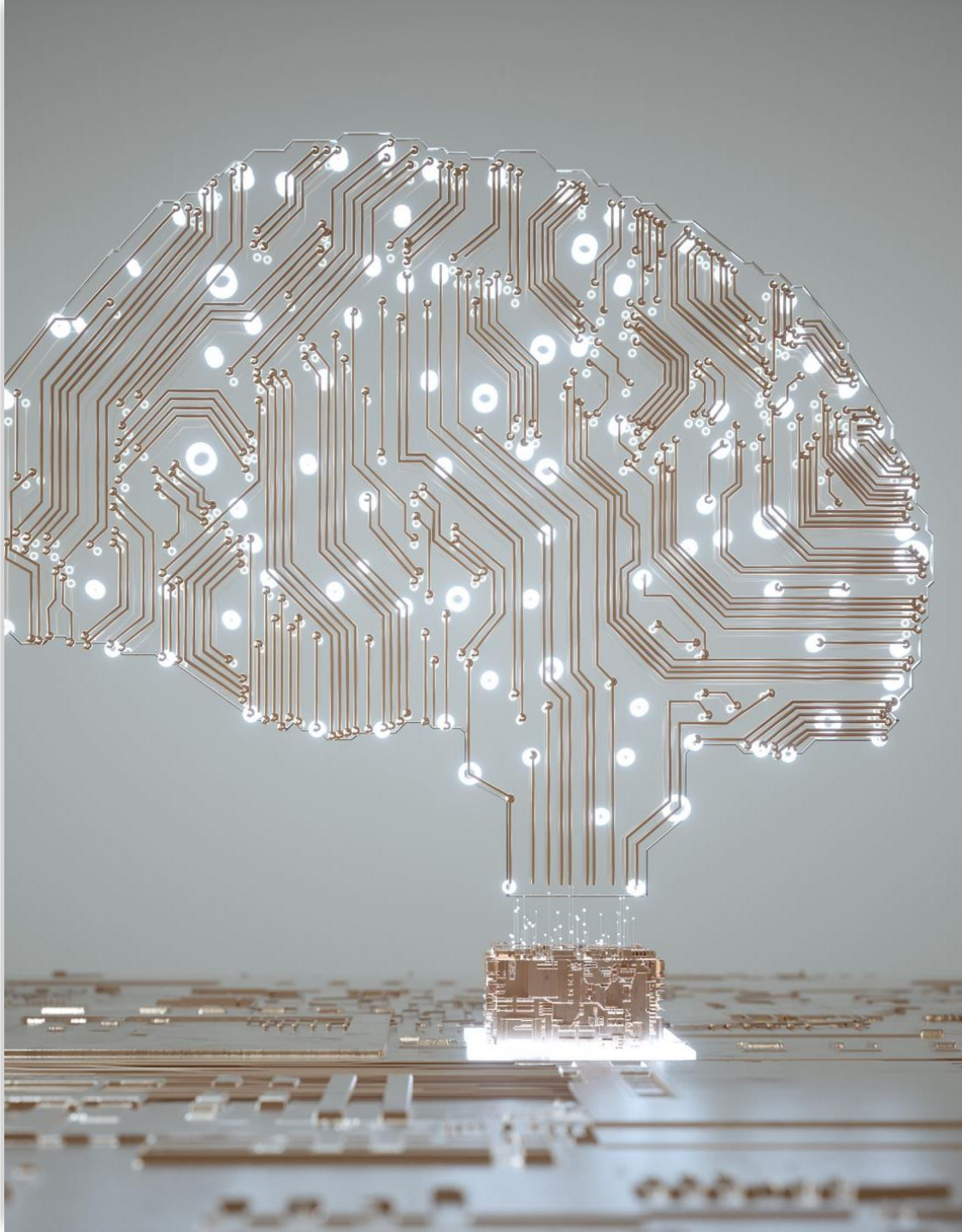
- **Processing Complex Rules**: Rule engines can handle complex, interconnected rules that might be challenging and time-consuming for humans to process manually. This capability to manage and reason through complex rule sets is a foundational aspect of AI.
- **Dynamic Decision Making**: Advanced rule 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

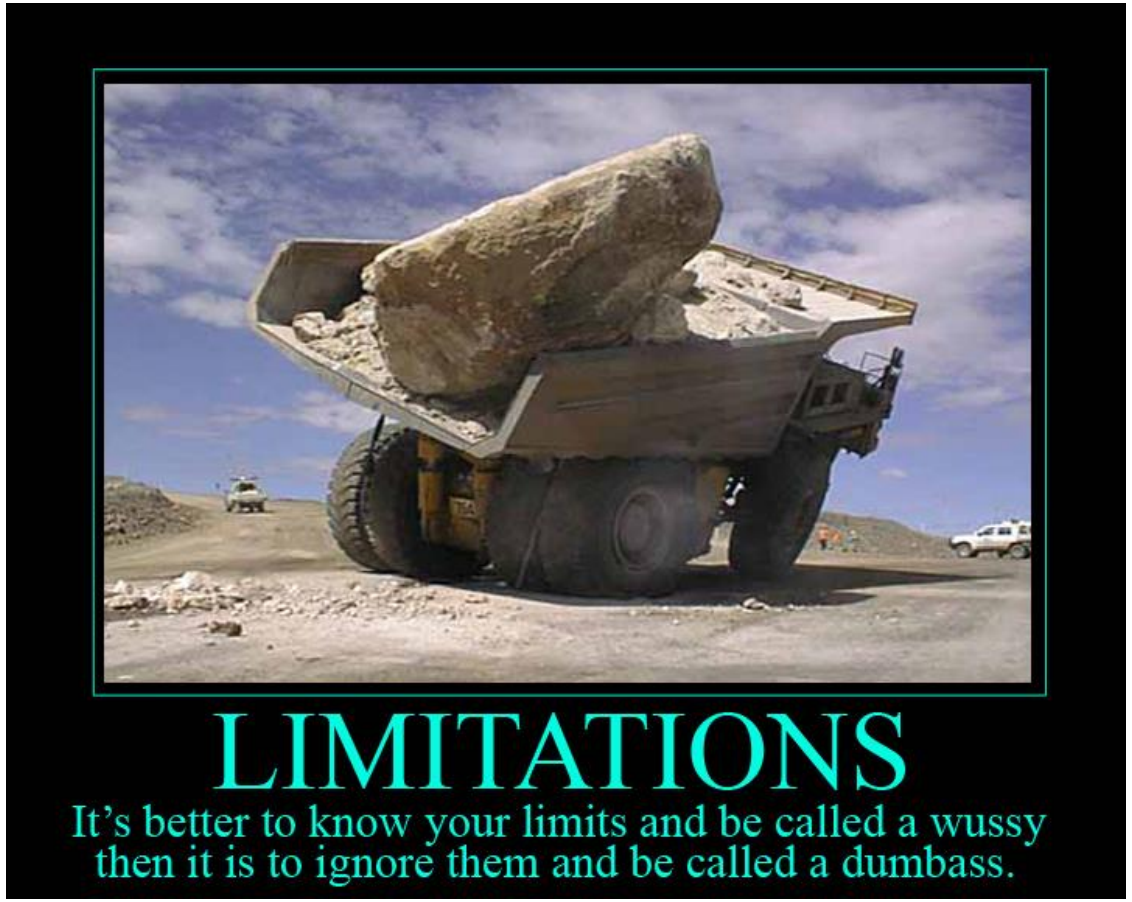
- **Handling Large Scale Data**: Rule engines can process vast amounts of data and make decisions much faster than a human could. This scalability is a key feature of AI, enabling efficient processing and analysis of large datasets.
- **Consistent Application of Rules**: Unlike humans, rule engines do 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.



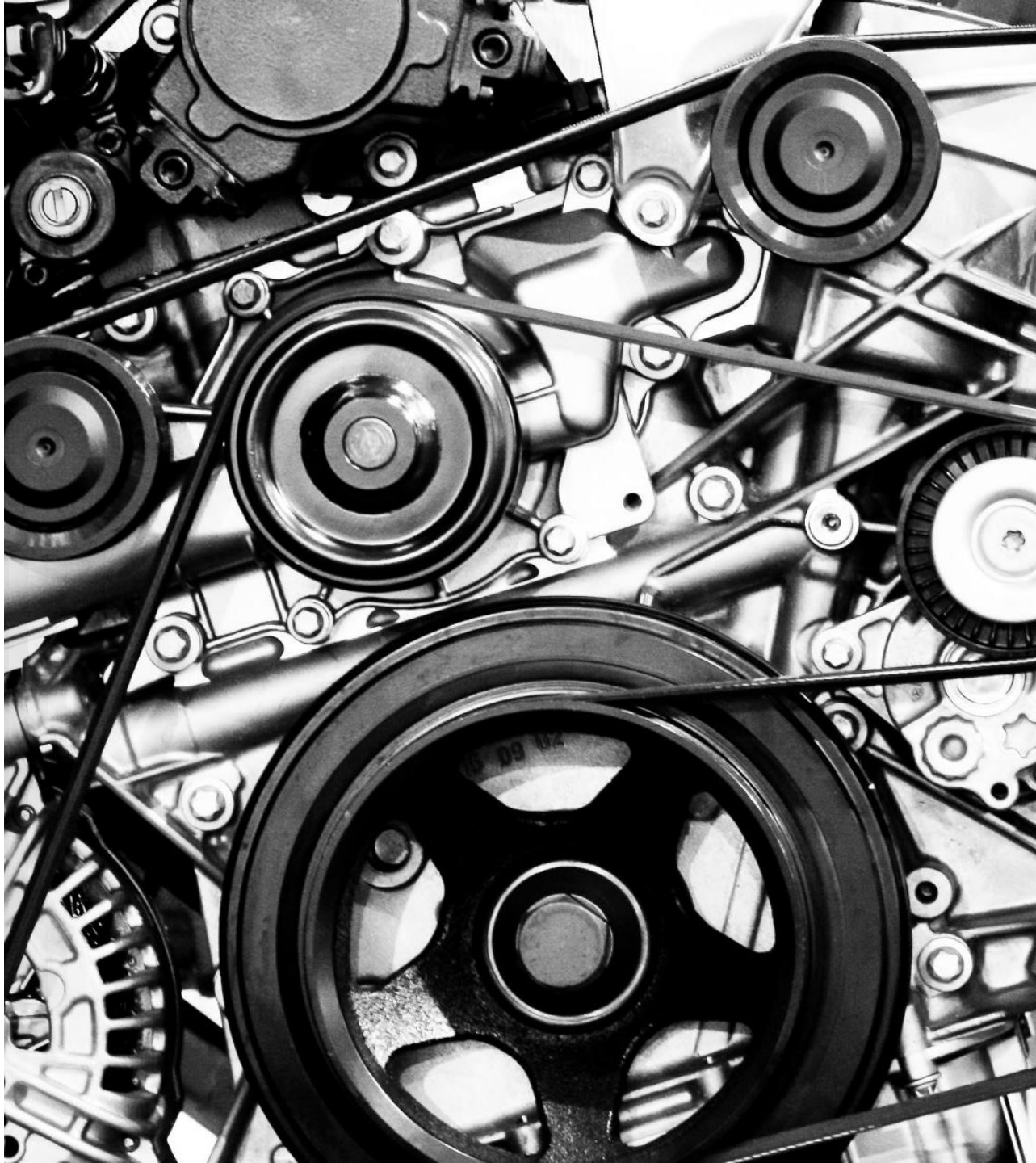
Rule Engine Components

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.

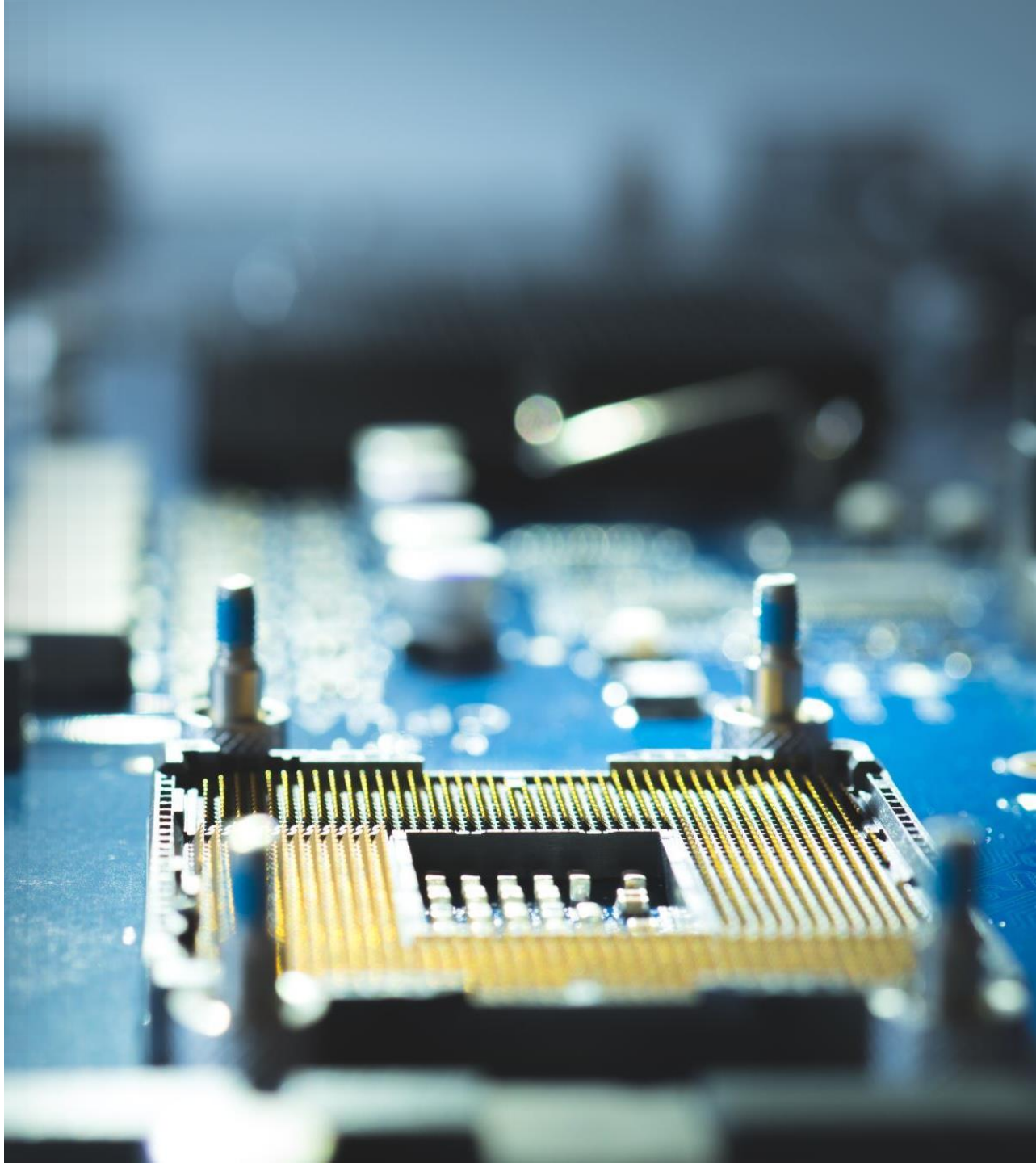
Rule Engine Components

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.



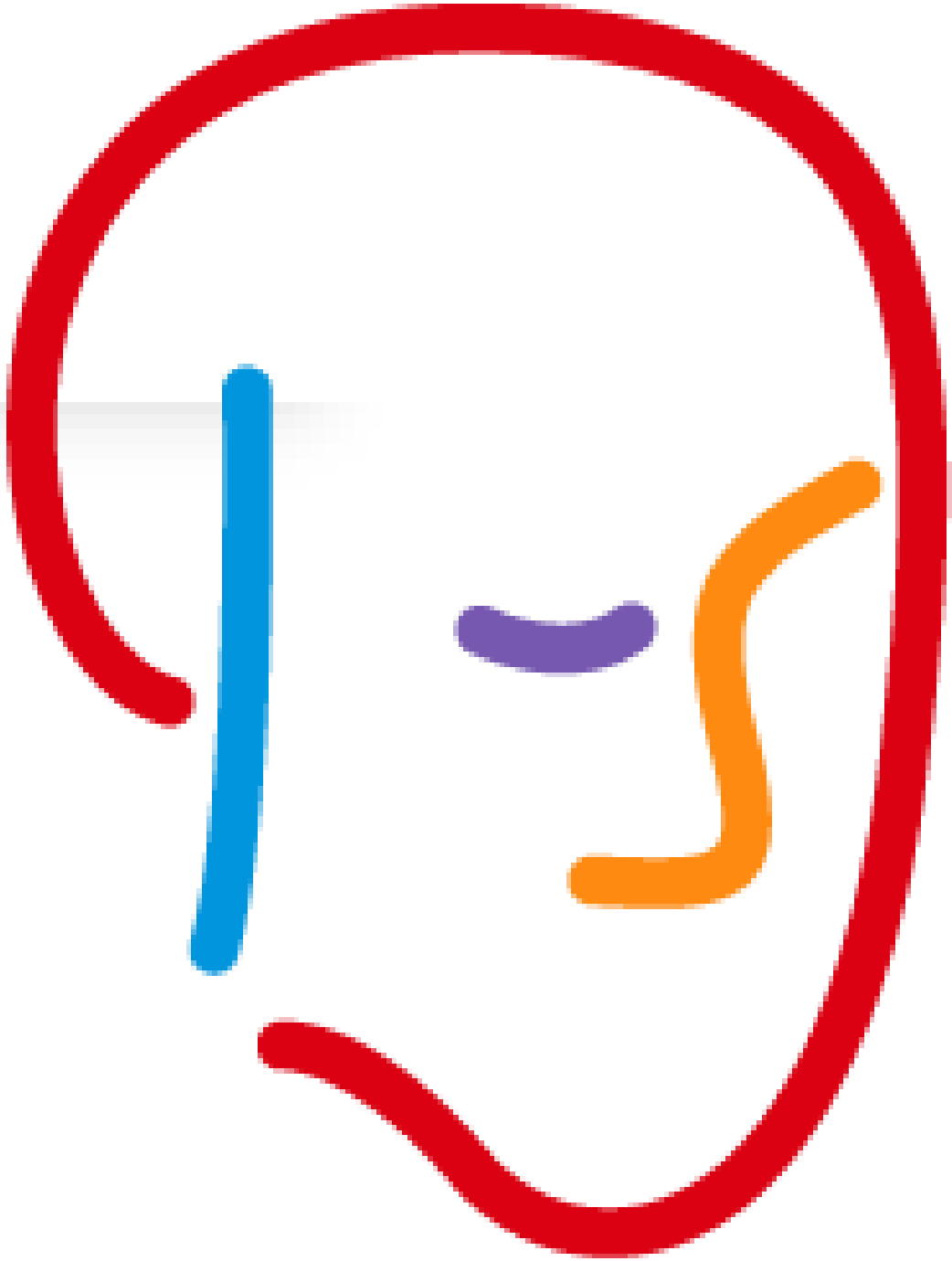
Rule Engine Components

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.

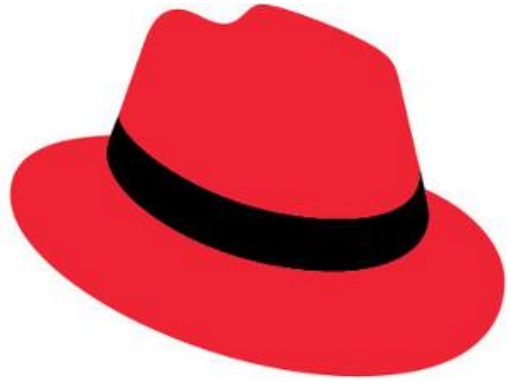
The project was named "Drools" as a play on the word "rules" and was developed as an open-source initiative from the start.

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.

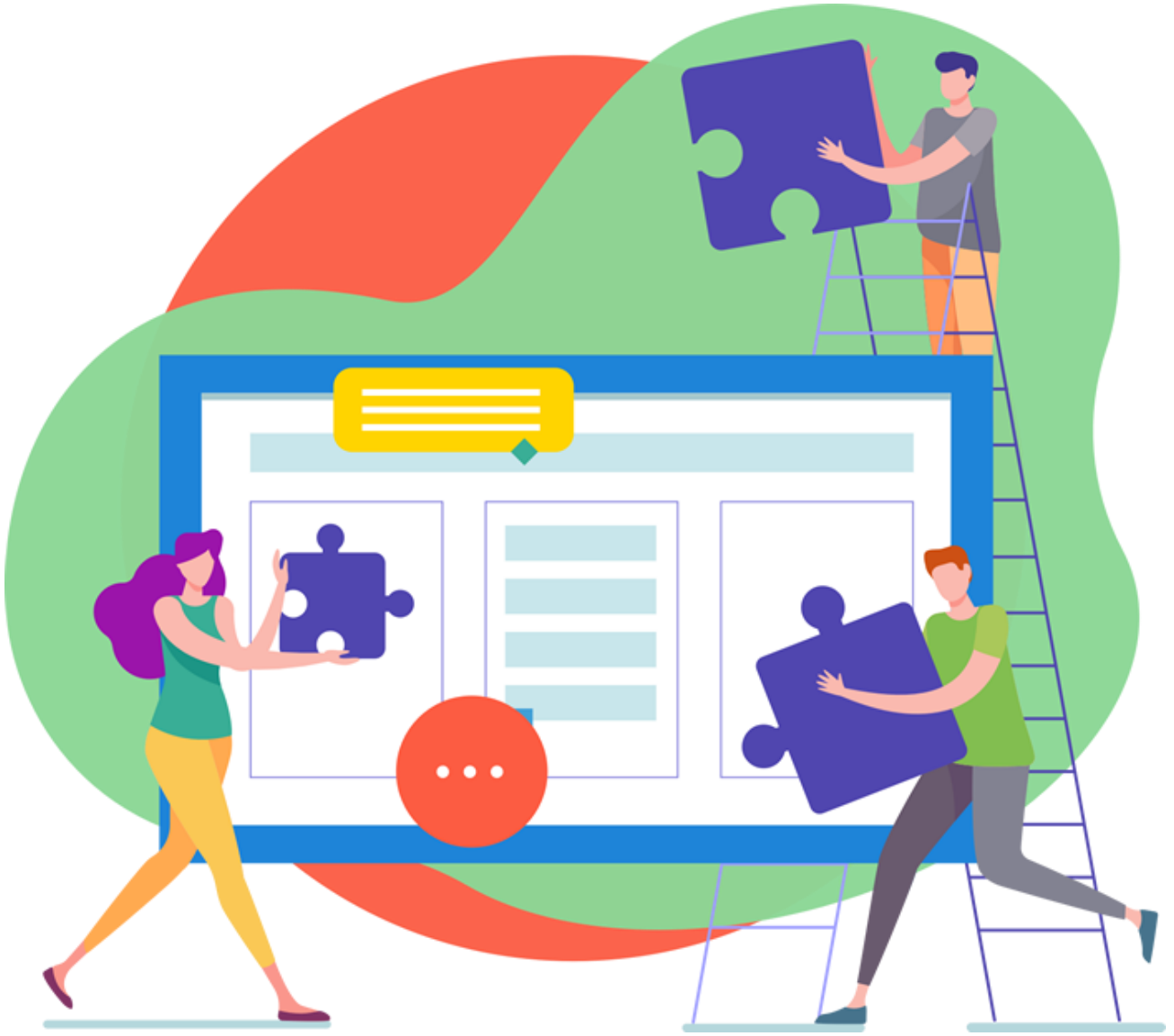
```
package com.example.rules;

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

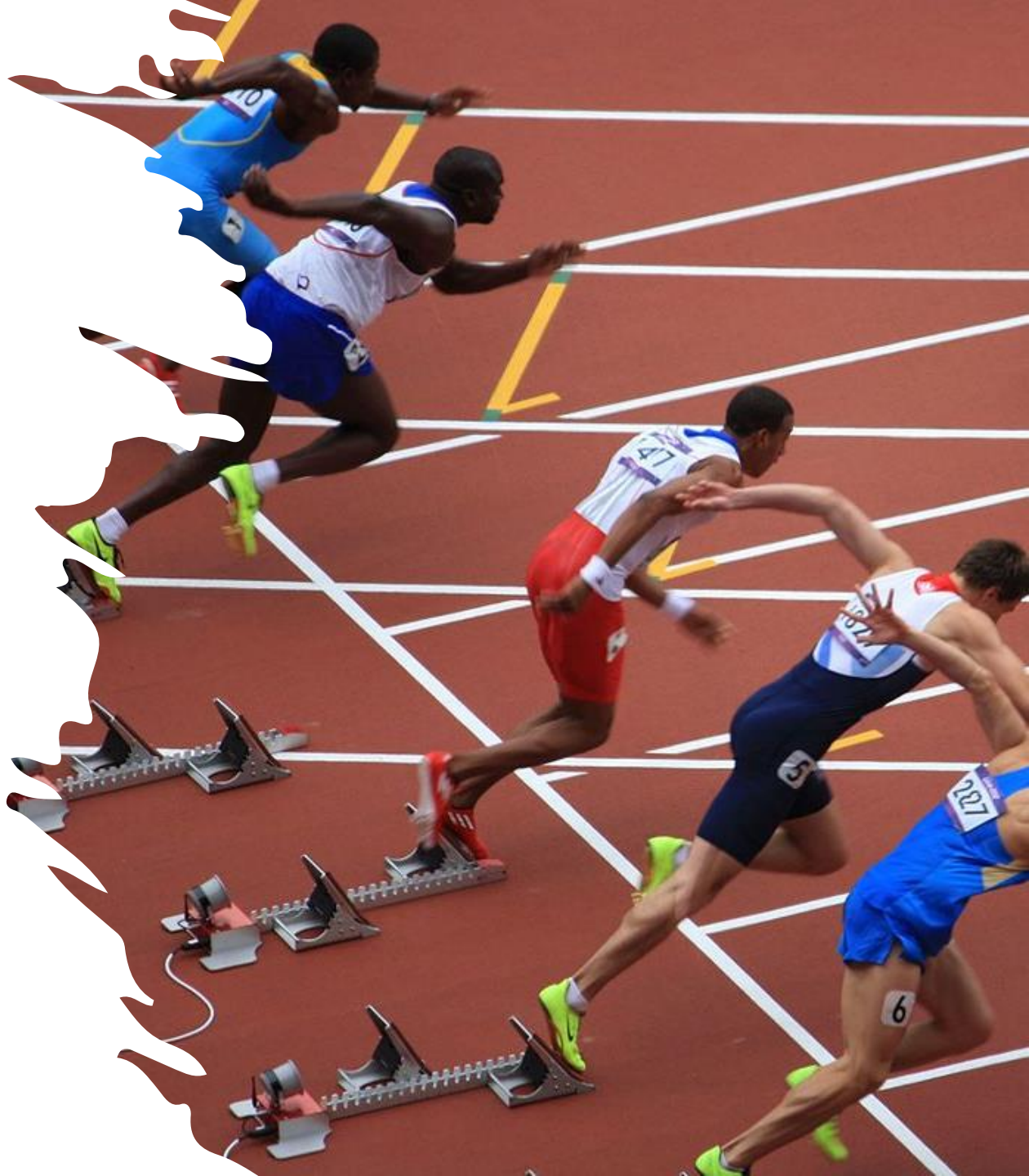
What to Expect? Through this exercise, you will:

- 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

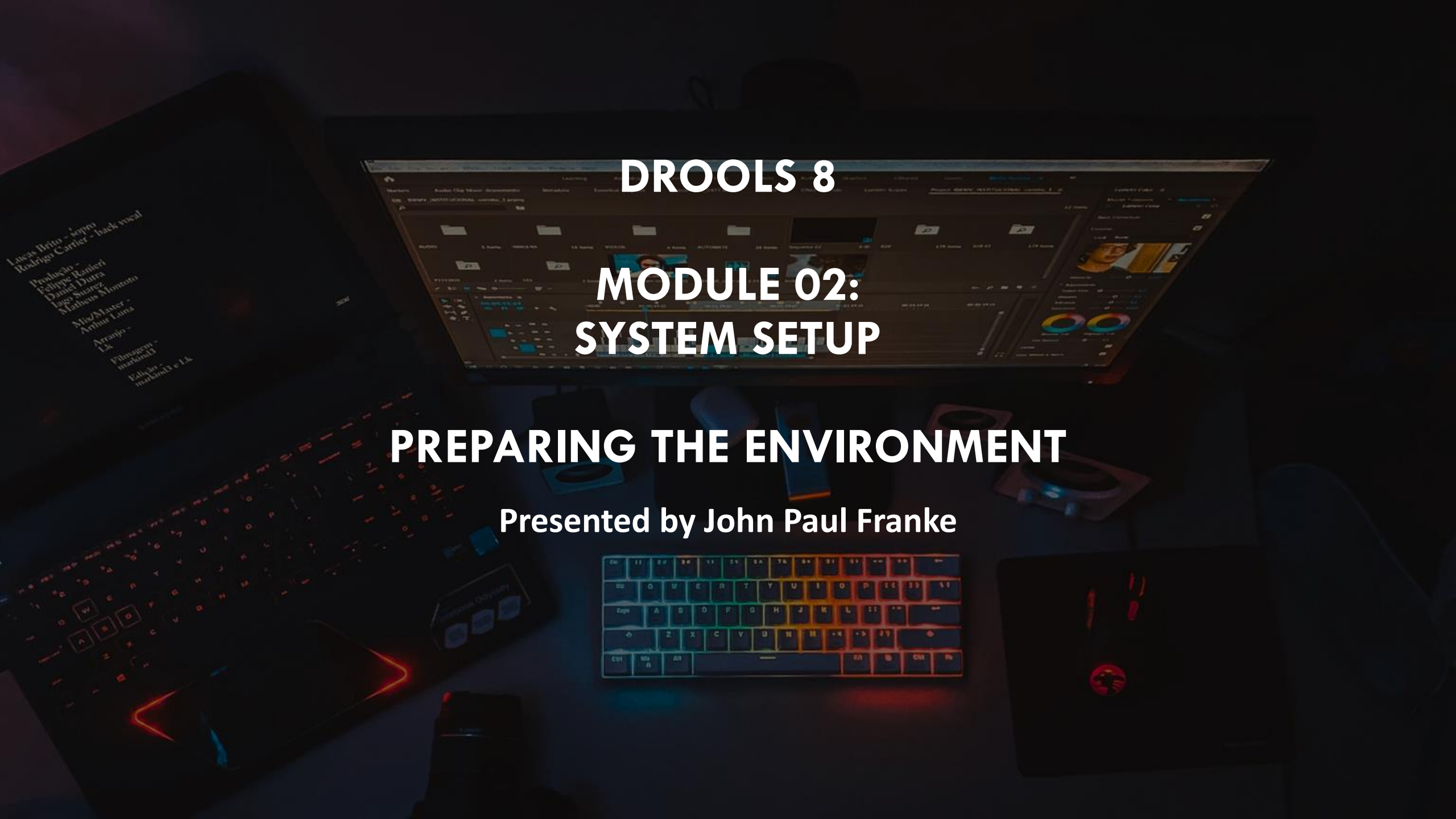


Quiz Time!

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.

Thank You!



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.

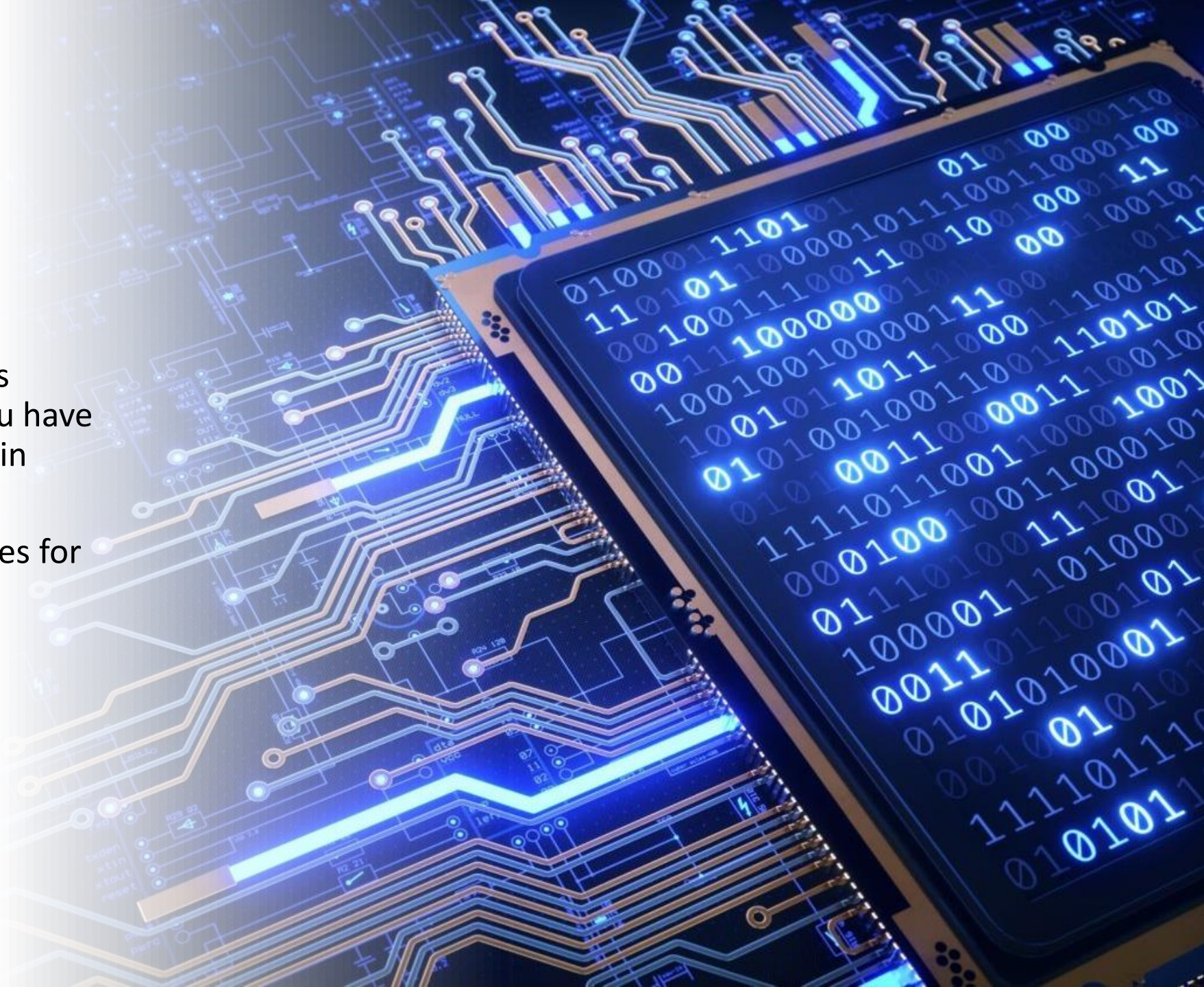
By the end of this module, you'll have a fully functional Drools development environment, ready to create and run rule-based applications. This setup is crucial as it lays the groundwork for all the exciting Drools features and projects we'll explore in the subsequent modules.

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

Drools Prerequisites

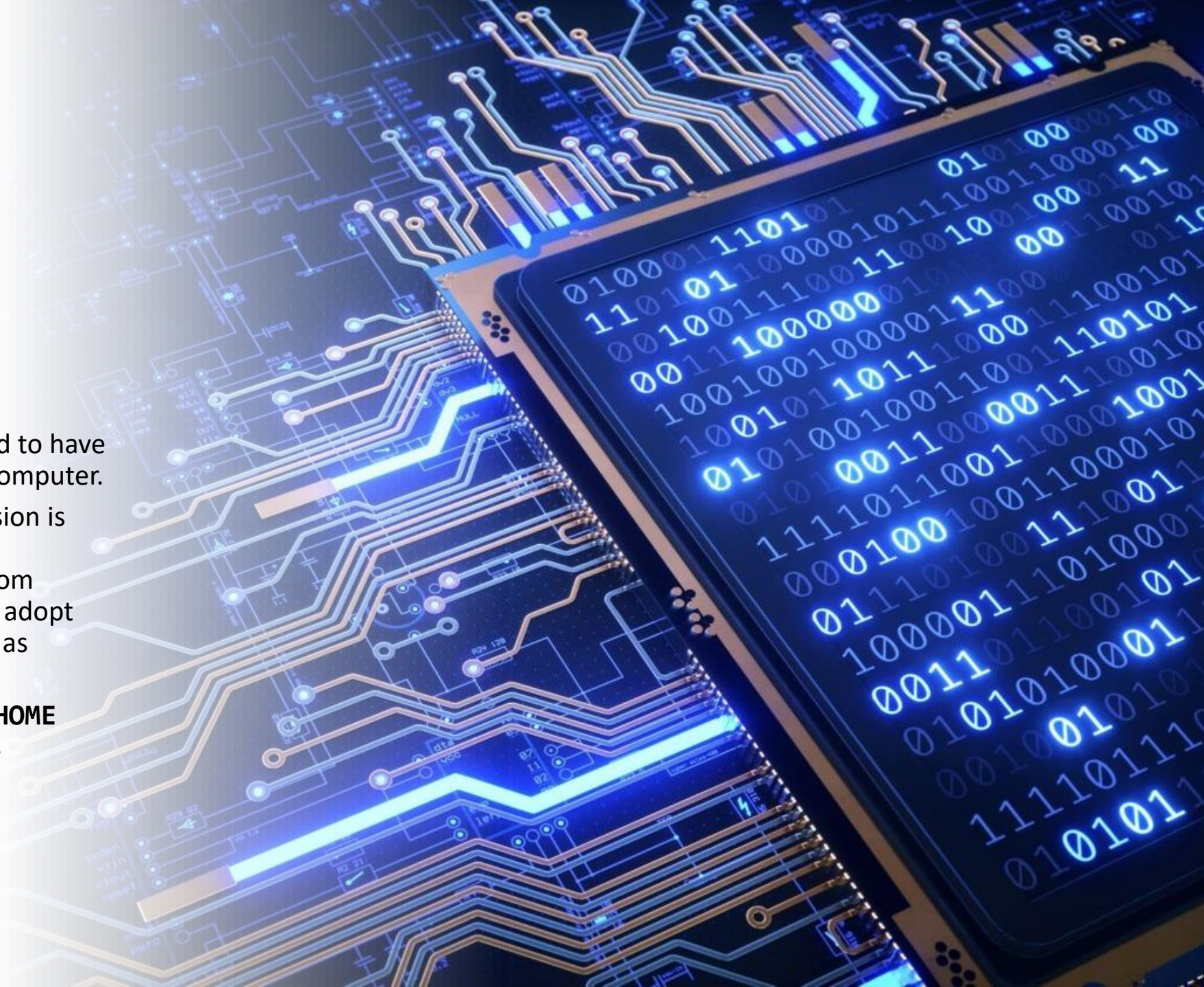
Before diving into Drools, it's important to ensure that you have the necessary prerequisites in place.

Here are the key prerequisites for working with Drools...



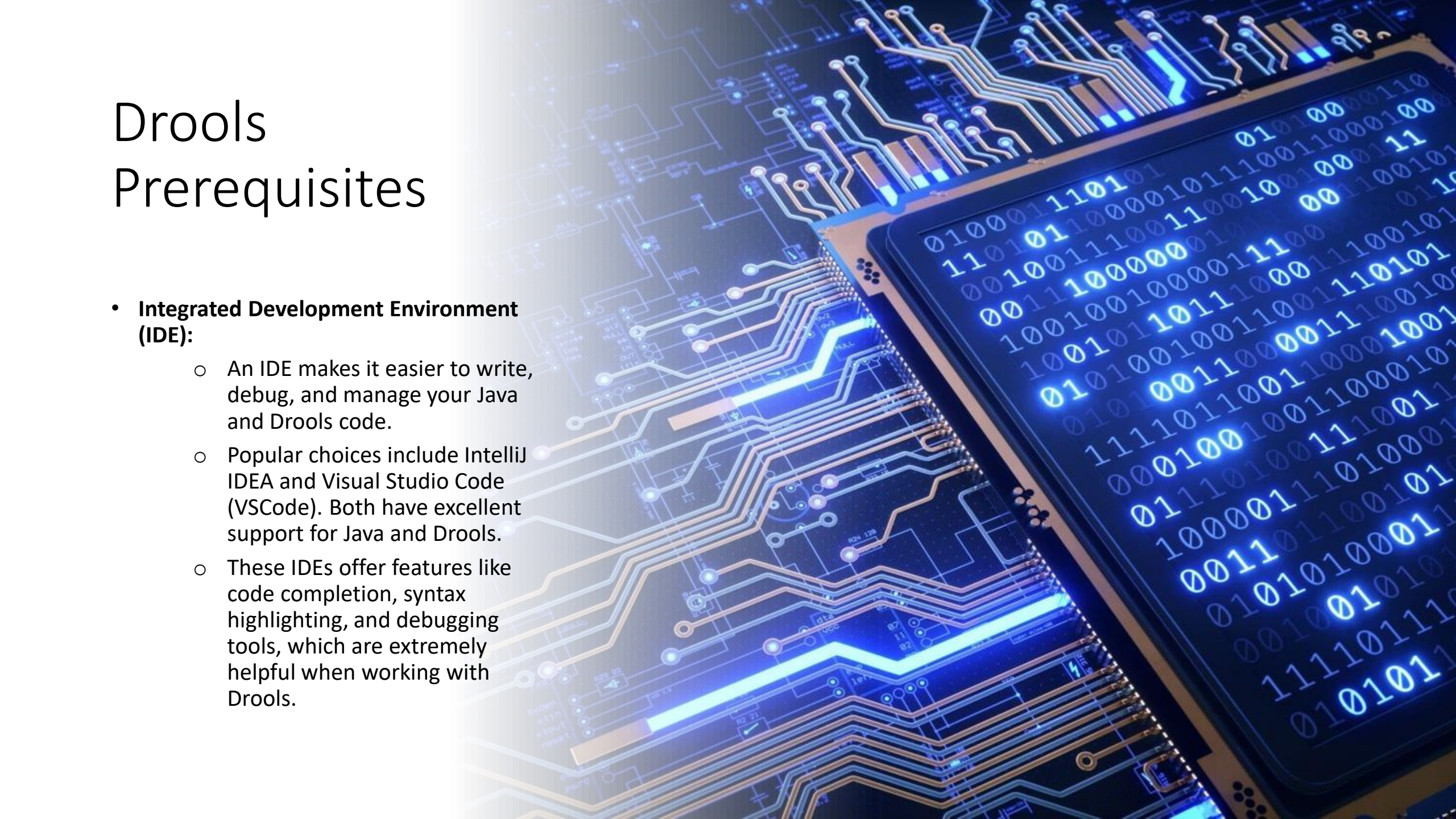
Drools Prerequisites

- **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.



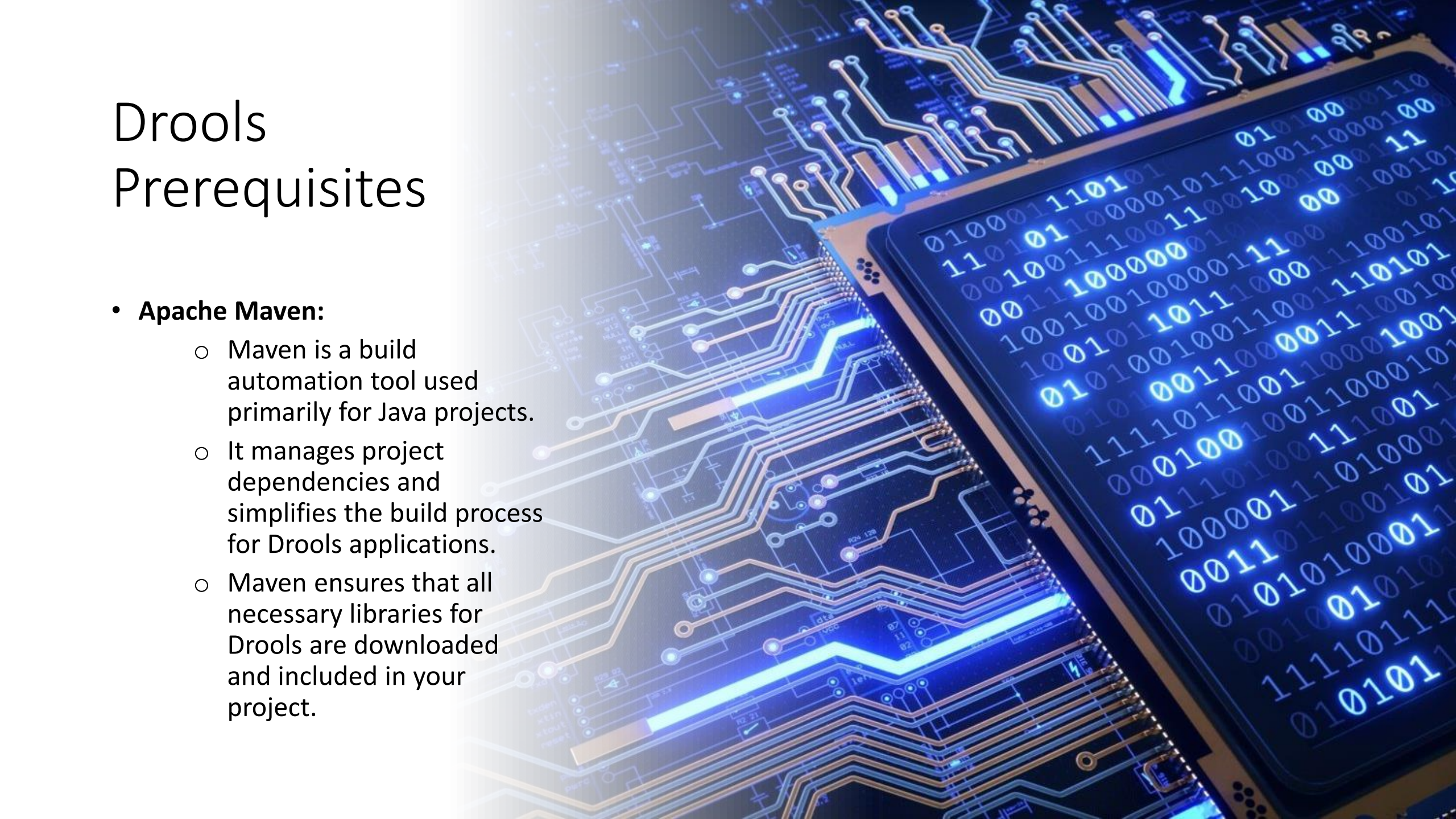
Drools Prerequisites

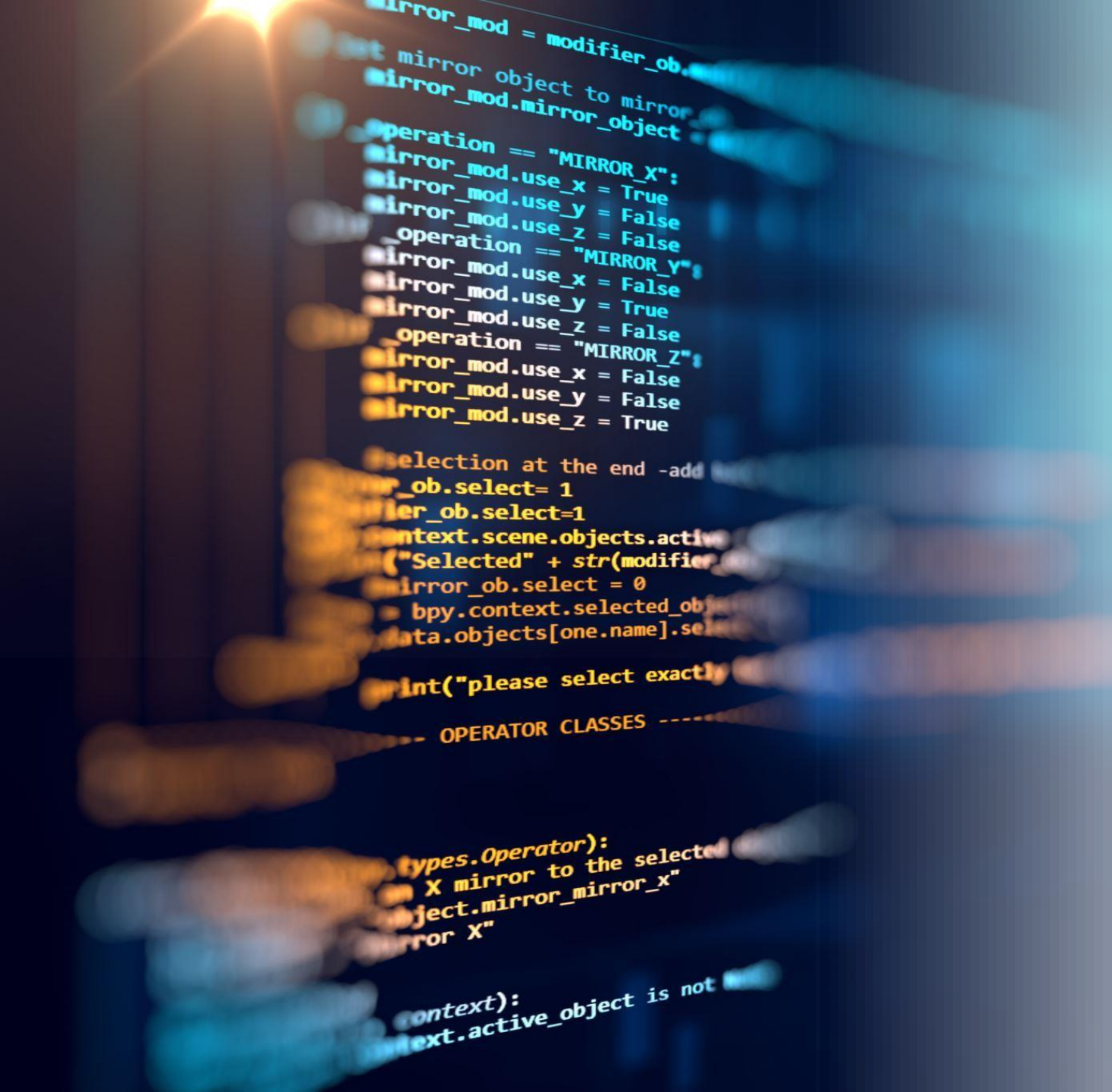
- **Integrated Development Environment (IDE):**
  - An IDE makes it easier to write, debug, and manage your Java and Drools code.
  - Popular choices include IntelliJ IDEA and Visual Studio Code (VSCode). Both have excellent support for Java and Drools.
  - These IDEs offer features like code completion,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



Drools Prerequisites

- **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

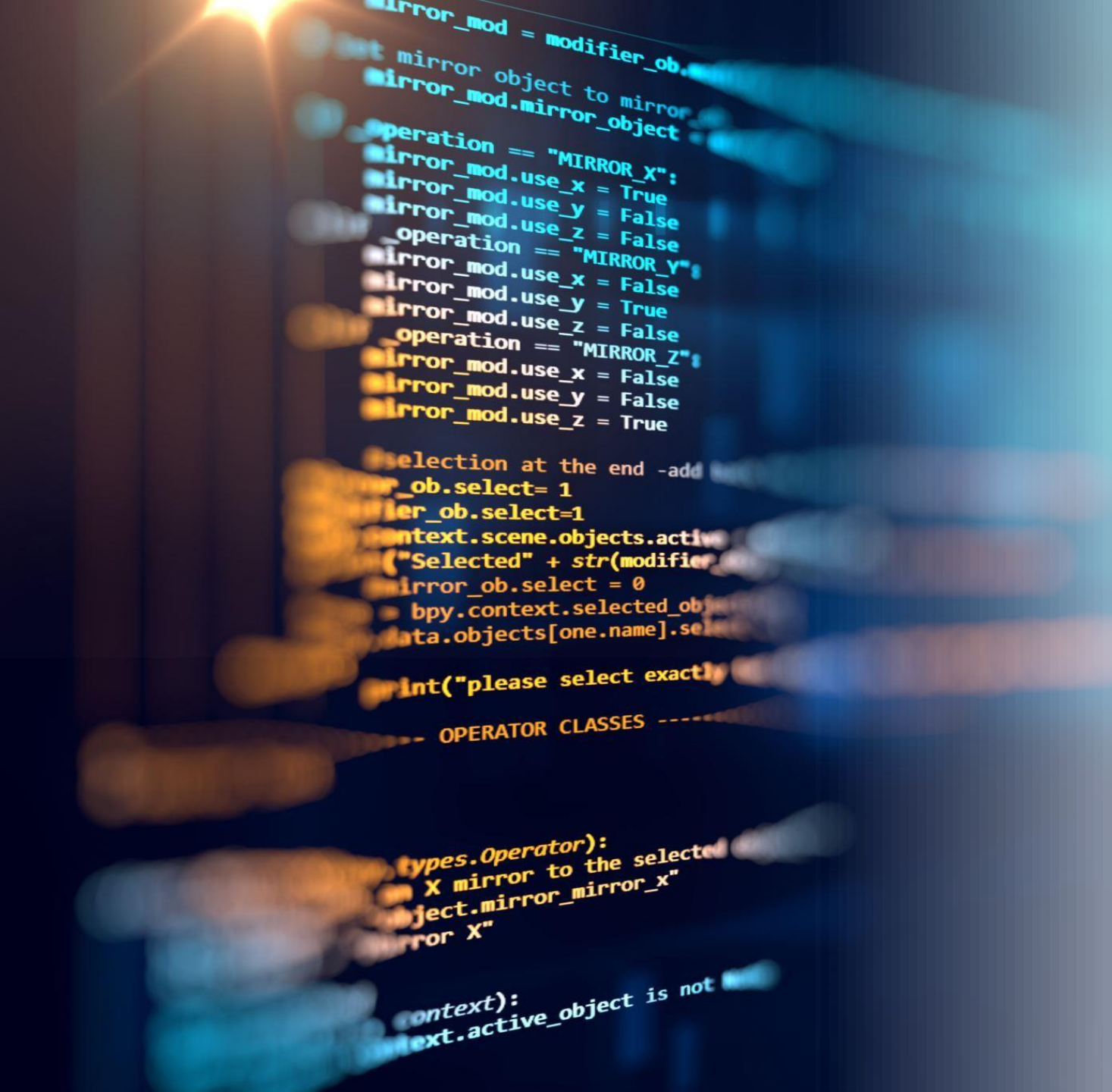
Prerequisites:

- 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.

Intro: Lab 02 – Java Installation

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:

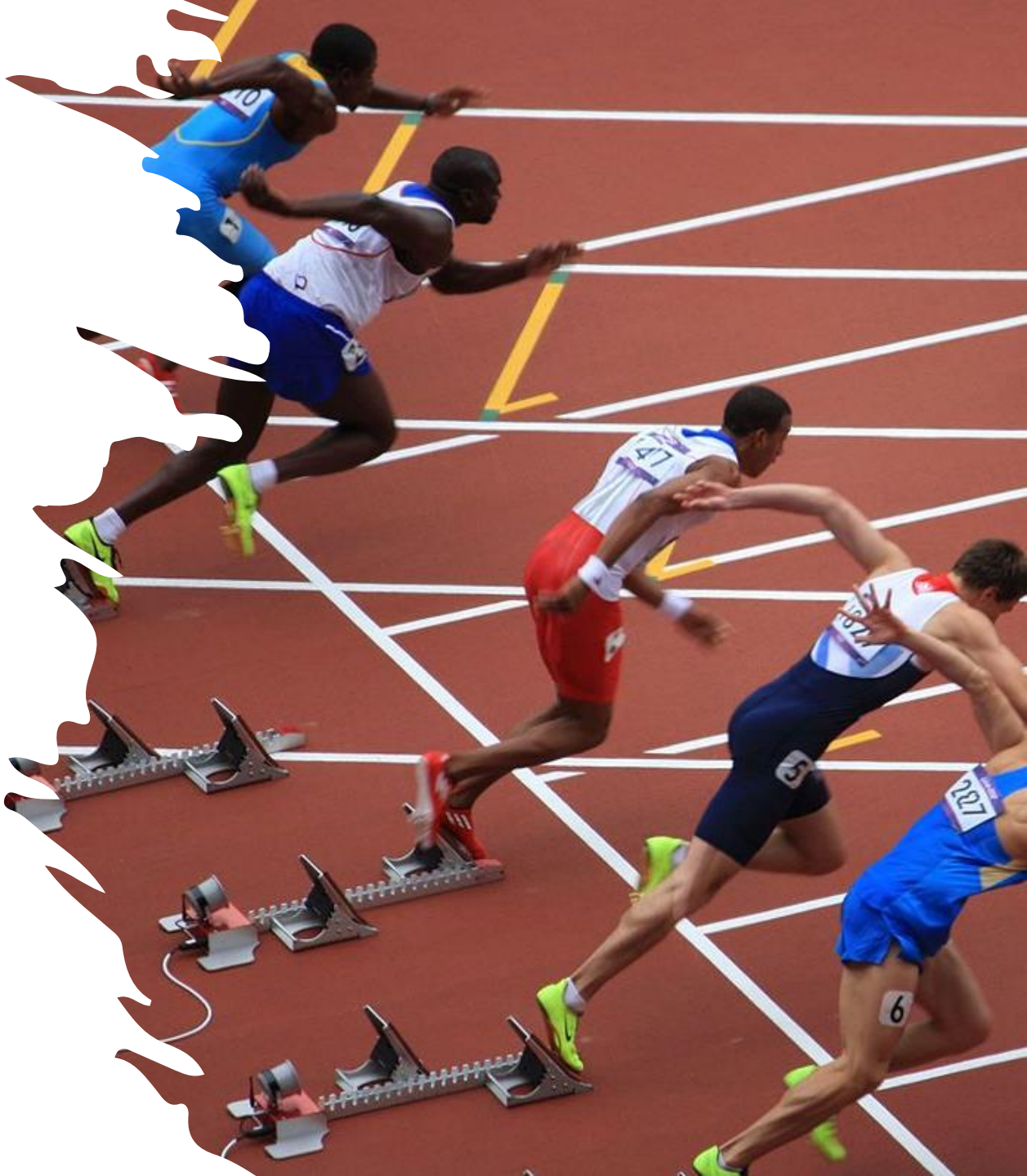
Our goal in this lab is to guide you through installing a compatible Java Development Kit (JDK), a critical component needed to run Drools.

Why Is This Important?

Without Java, Drools cannot run. This lab will prepare your system for running the labs and materials for the rest of the course

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 - installing Apache Maven.

Objective: In this lab, we will install and verify Apache Maven on your system. Maven is a powerful tool for project management and build automation, is needed 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.

Intro: Lab 03 – Maven Installation

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?

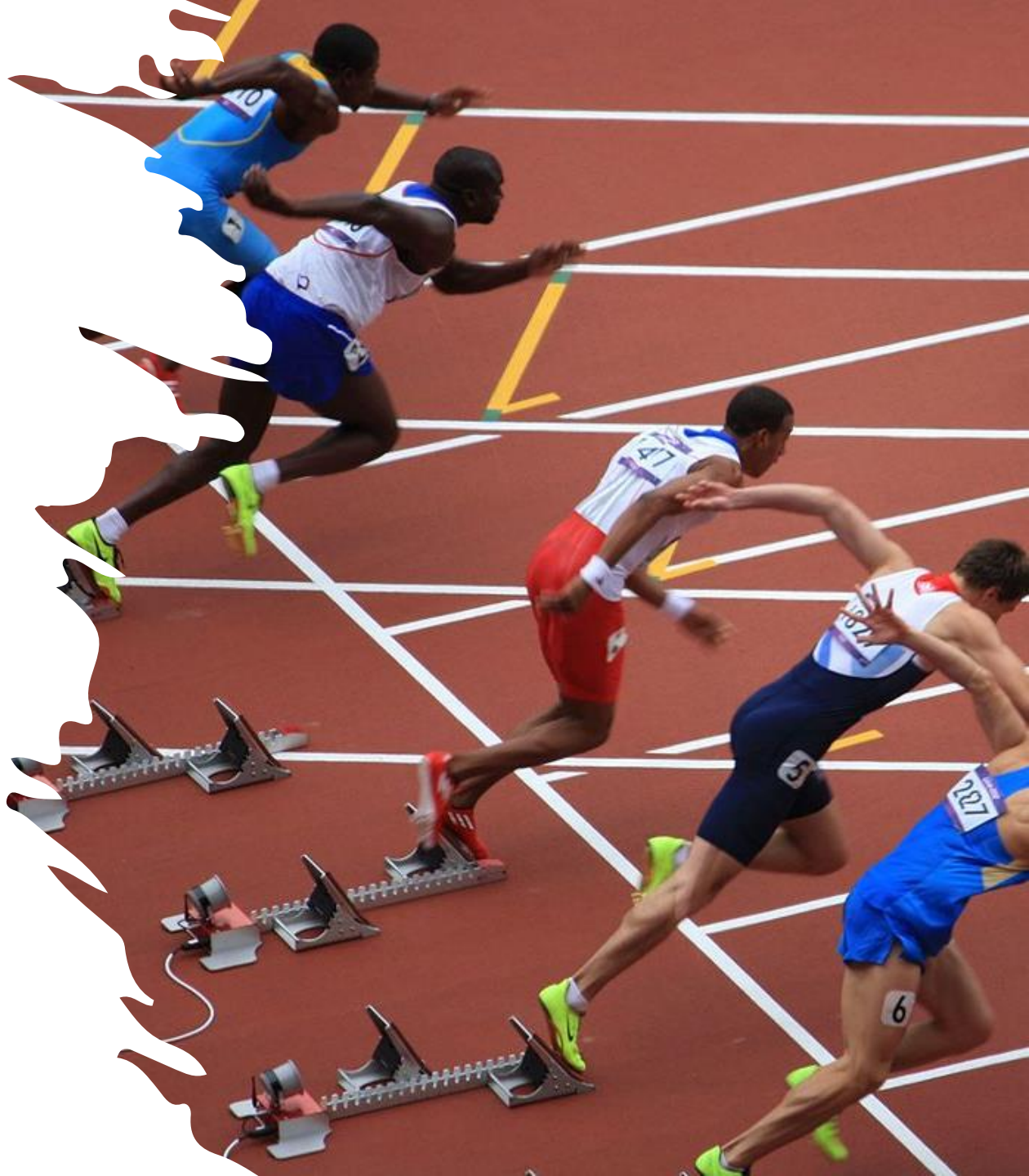
Maven simplifies the management of Java projects by automating tasks like building, reporting, and documentation. For Drools, Maven helps in efficiently handling project dependencies, ensuring that all necessary libraries and tools are in place for your Drools projects.

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.

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.

Intro: Lab 04 – IntelliJ IDEA Installation

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?

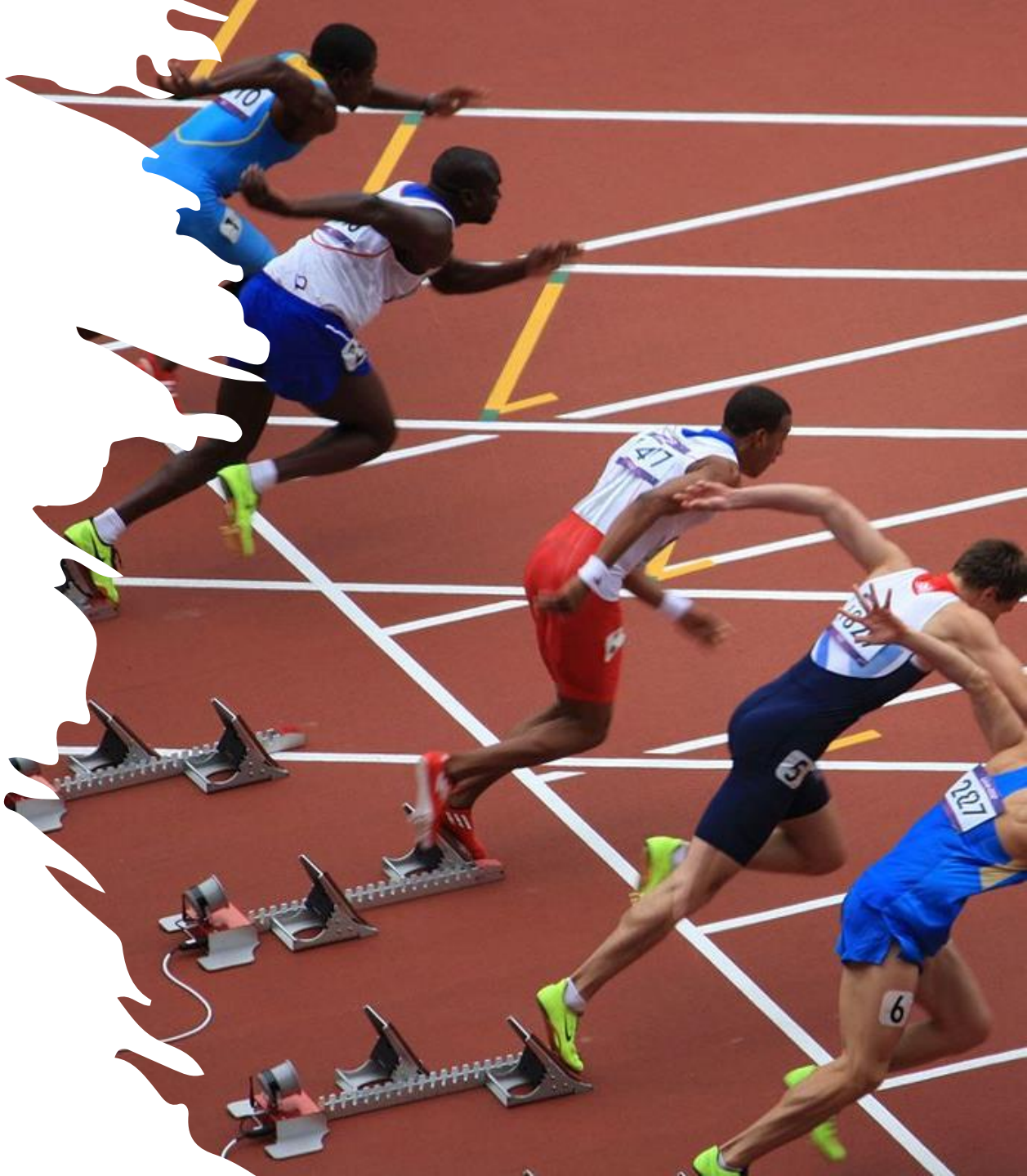
IntelliJ IDEA is renowned for its robust set of features, including intelligent coding assistance, ergonomic design, and powerful tools for Java developers. It's particularly advantageous for Drools development due to its advanced code navigation and debugging capabilities.

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



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.

Quiz Time!

Thank You!

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. 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!

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.

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.

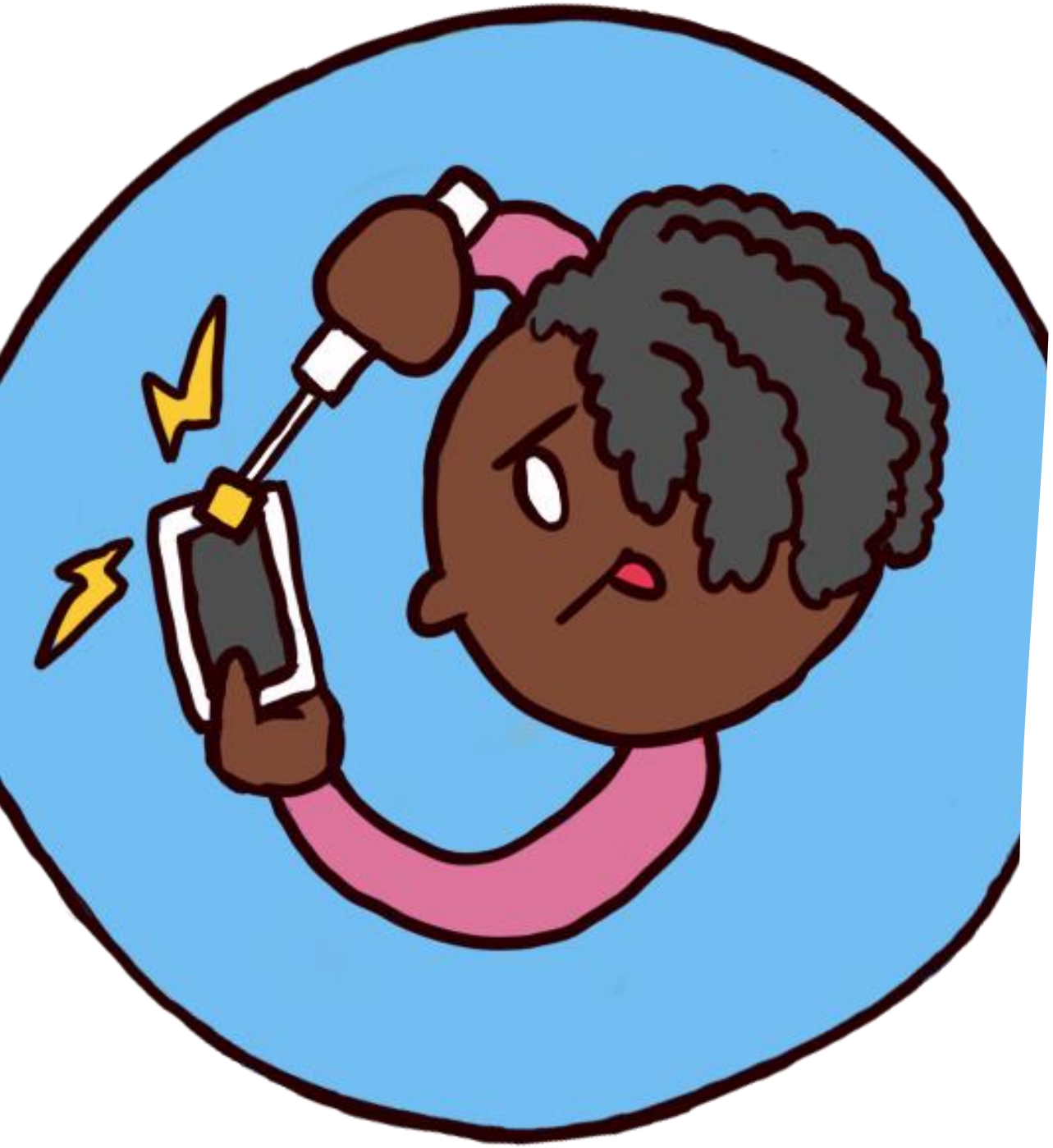
Understanding KIE Sessions

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.

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.

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.

Intro: Lab 05 – Drools Project Setup

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.

Intro: Lab 05 – Drools Project Setup

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.

Intro: Lab 06 – Drools Setup with VSCode

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.

Intro: Lab 06 – Drools Setup with VSCode

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.

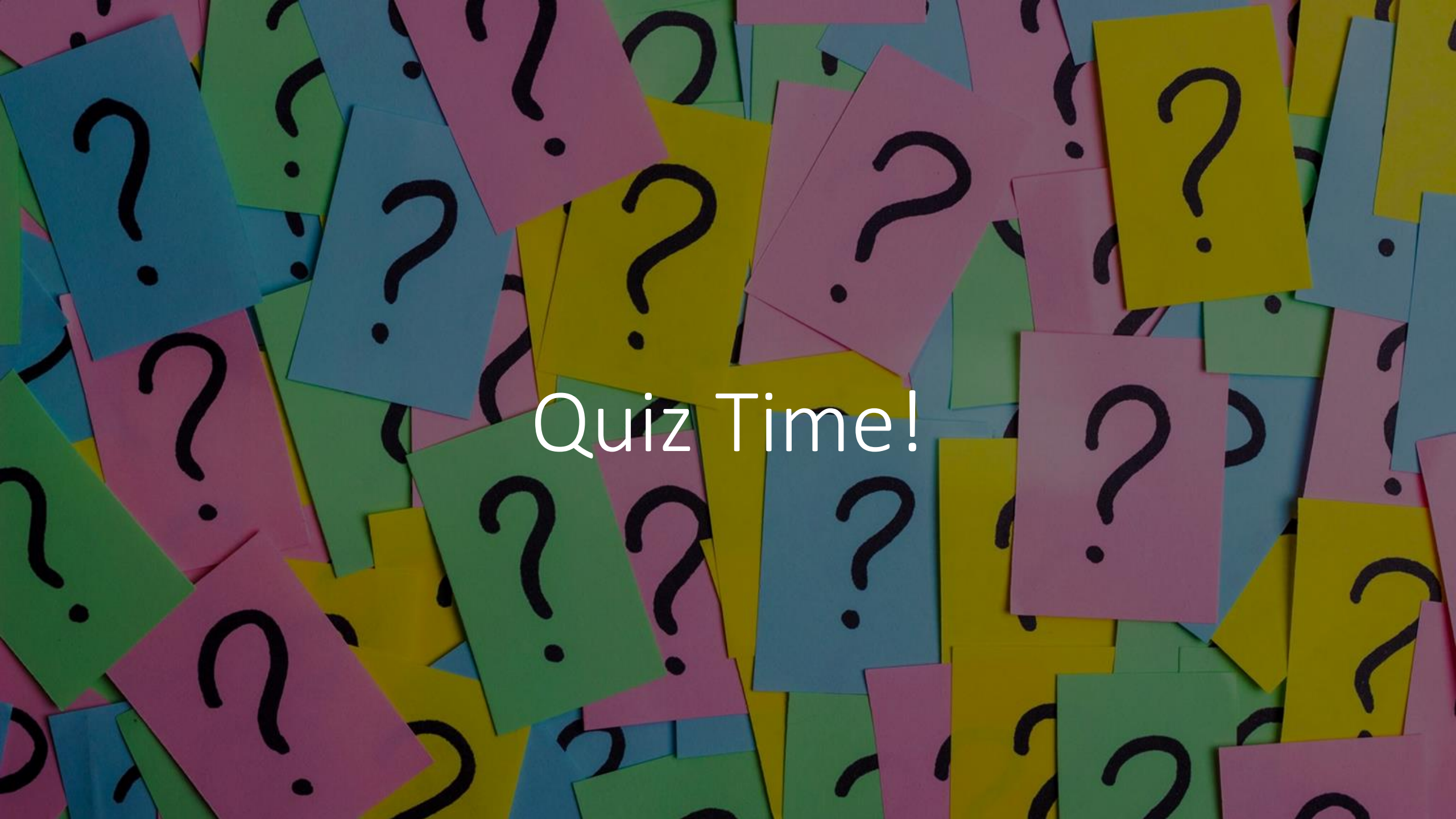
Intro: Lab 06 – Drools Setup with VSCode

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 fast setup and customizable experience, while still providing powerful features for Java and Drools development.

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!

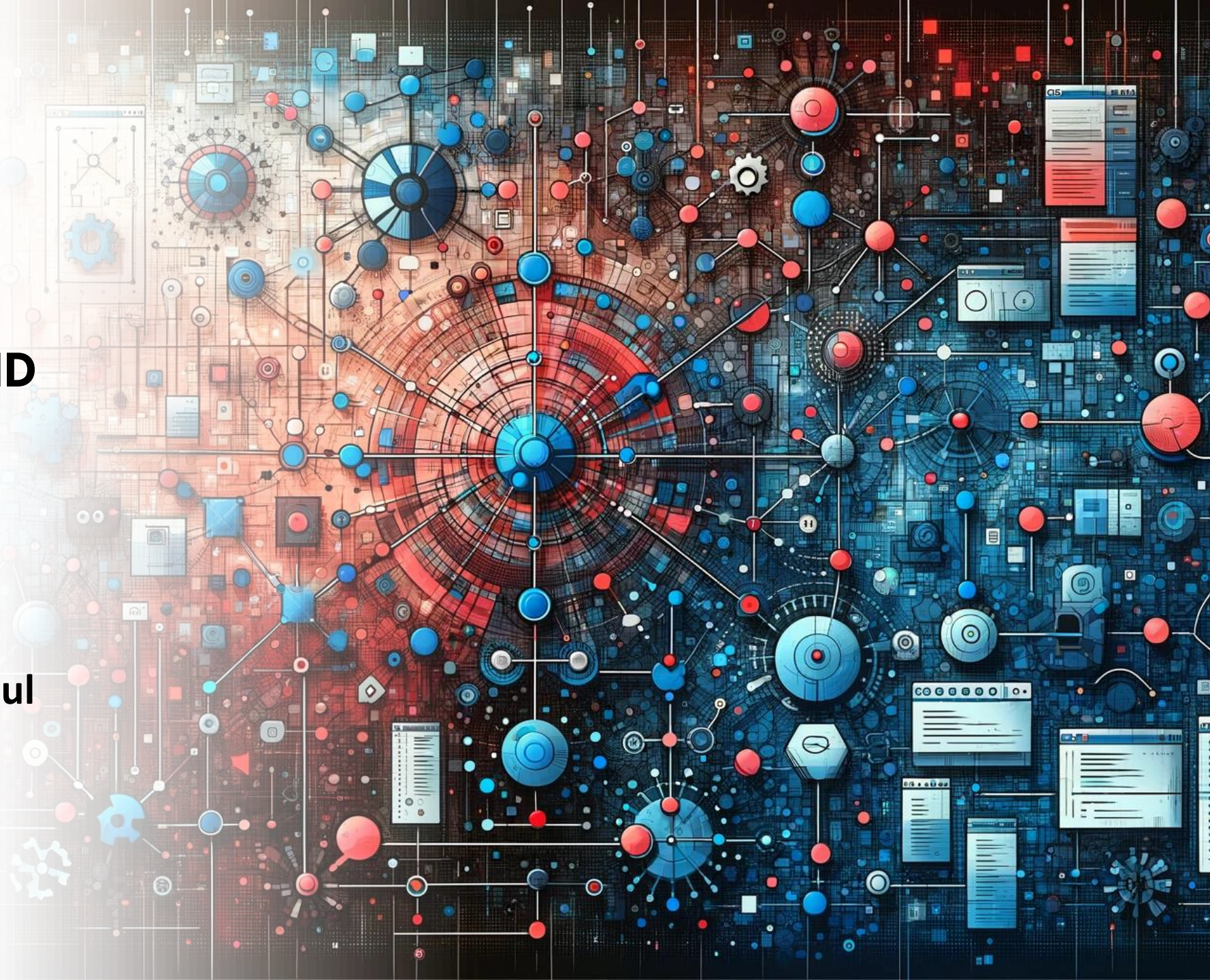
Thank You!

DROOLS 8

MODULE 04: KIE SESSIONS AND RULE LOGIC

CREATING AND RUNNING RULES

Presented by John Paul Franke



Learning Objectives

Here are our learning goals for this Module:

- Understand basic rule creation
- Learn to use logical operators within rules.
- Explore variable binding and usage for detailed reporting.
- Discover how to modify fact object properties using rule variables.
- Understand the concept of inference and the limitations of stateless sessions.
- Use the `modify` keyword to track changes within a session.
- Grasp cross-product checks between different fact types.
- Learn strategies to eliminate unnecessary rule firing and enhance efficiency.

Case Study – Health Insurance

Our health insurance case study, incorporated into an interactive lab environment, is designed to give you a hands-on experience with Drools, enabling you to see firsthand how business rules can be used to automate complex decision-making processes in a real-world scenario.

We will examine the Base Objects and fact objects that comprise the 'facts' or 'data' of our scenario, see how these objects interact with rules, and how they all come together in active KIE sessions.

Case Study – Interactive Environment

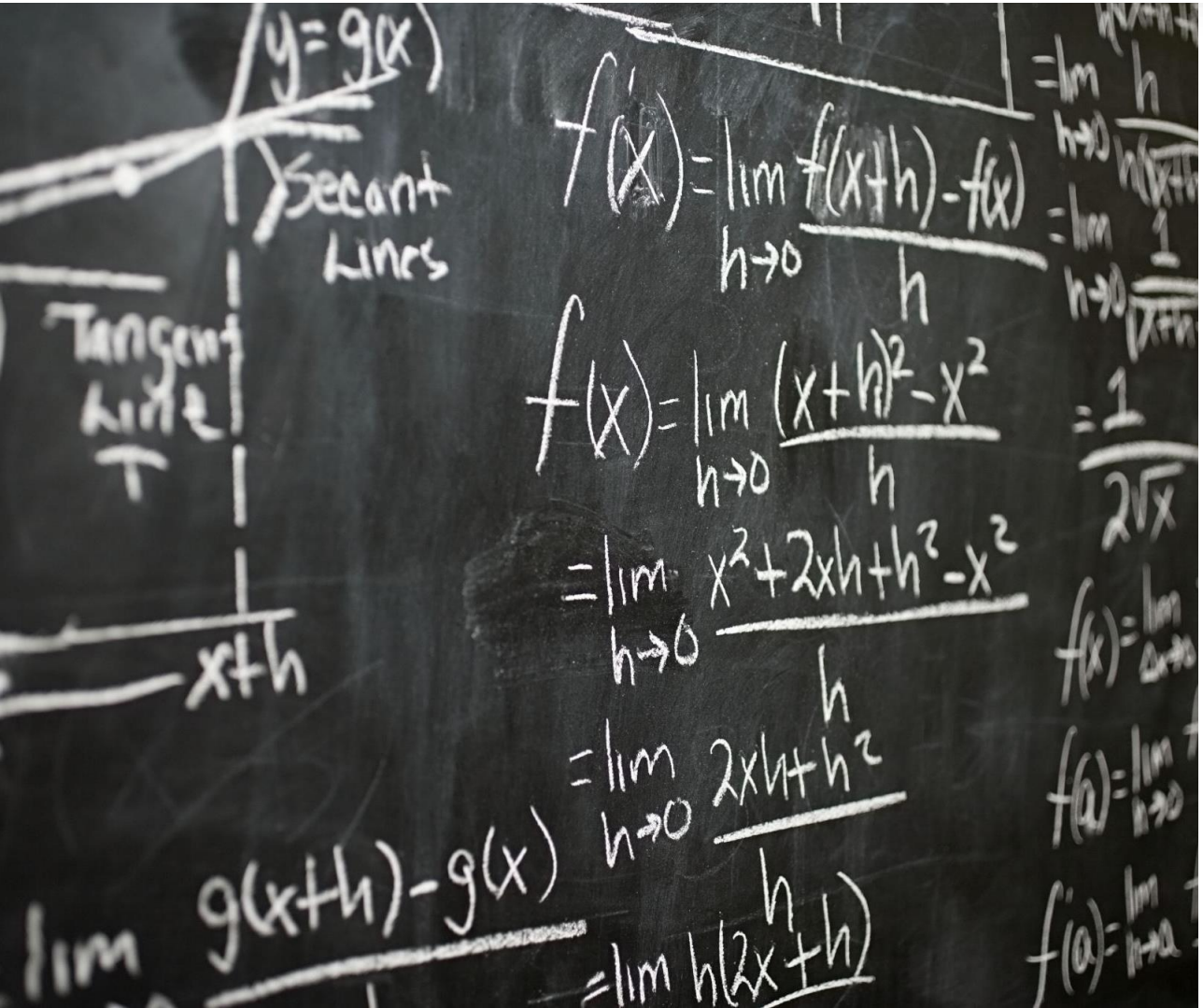
You will also have the chance to get hands-on, alter the environment, create new rules, objects, and sessions, and put your knowledge to practice with lab exercises and challenges.

Case Study – First Facts

Meet your clients – these fine folks have submitted health insurance policy applications and have submitted their data, which is represented as attributes in our `Application.java` class. We can base our rules on this information. This is how their info would look as a data table:

Application Number	Client Name	Client Age	P.E.C.	Risk	Employed	BMI	Client ID Number	Smoker
A-1000	James Smith	52	False	High	True	21	ID-100	False
A-1001	Maria Smith	56	True	Medium	True	23	ID-101	True
A-1002	Robert Johnson	38	False	High	True	25	ID-102	True
A-1003	Linda Johnson	36	True	Low	False	32	ID-103	False
A-1004	Michael Johnson	16	False	Low	False	20	ID-104	False
A-1005	Elizabeth Anderson	27	False	Medium	True	27	ID-105	True
A-1006	David Lrr	63	True	High	True	31	ID-106	True
A-1007	Emily Taylor	80	False	Low	False	18	ID-107	True
A-1008	William Brown	21	True	Low	True	28	ID-108	True
A-1009	Jessica Thompson	71	False	Medium	False	23	ID-110	False

DRL – Basic Operators



Comparison Operators

- **==** (equals)
- **!=** (not equals)
- **<** (less than)
- **<=** (less than or equal to)
- **>** (greater than)
- **>=** (greater than or equal to)

Logical Operators

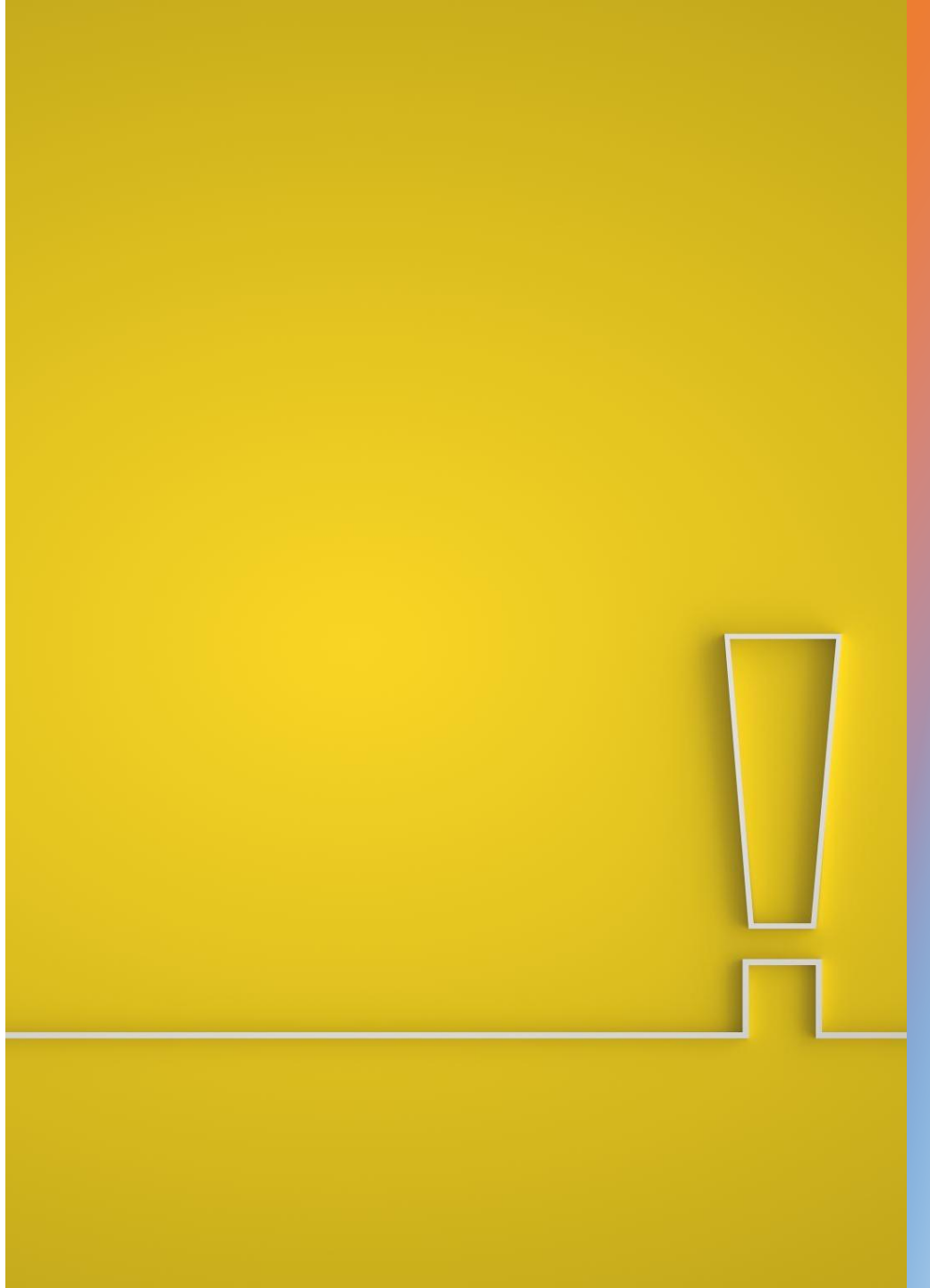
- **&&** (logical AND)
- , (alternative AND)
- **||** (logical OR)
- **not** (negation)

Case Study – Lesson 01: Stateless Sessions

Stateless Knowledge Sessions in Drools are used when there is no requirement for a rule engine session to maintain state over time. This makes it ideal for scenarios where each execution of the rules is independent of previous or future executions.

Here are some common use-cases for stateless KIE (Knowledge Is Everything) sessions in Drools:

- **Single Transaction Scenarios**: In applications where rules are evaluated in a single transaction or operation, without the need for retaining session state across transactions.
- **Stateless Computations**: For scenarios where the rules do not rely on the previous state of the session, making each invocation independent.
- **Batch Processing**: Useful in batch processing jobs where a large number of business rule transactions are processed in a single operation, and there is no need to keep state between individual transactions.
- **Rule-Based Calculations**: For performing calculations or data manipulations where the outcome is determined solely by the current set of facts and rules, and prior states are irrelevant.
- **Content Routing**: For applications that use rules to route content or messages based on their content, without requiring knowledge of past messages.



Case Study – Lesson 01

Let's jump into our health insurance Case Study lab environment and explore basic rule creation and the running/execution of stateless sessions

Inference

"Inference" refers to the ability of a rule engine to deduce or conclude new information based on the existing set of facts and rules without being explicitly programmed to do so. This feature allows Drools to dynamically adapt its decision-making process as new facts are introduced or existing facts are modified

Inference - example

In this example, when an Applicant fact indicating the applicant is a smoker is inserted or updated in the working memory, the first rule fires and modifies the Applicant fact to set its riskLevel to "High". This modification triggers the second rule, which infers that the applicant is not eligible for premium insurance based on their high-risk level. The system inferred additional information (eligibility) based on the rules and facts available.

```
// Rule to infer high risk for smokers
rule "Identify High Risk Applicant"
when
    $applicant: Applicant(smoker == true)
then
    $applicant.setRiskLevel("High");
    update($applicant);
end
```

```
// Rule to infer eligibility based on risk level
rule "Infer Eligibility for High Risk"
when
    $applicant: Applicant(riskLevel == "High")
then
    System.out.println($applicant.getName() + " is not eligible for premium insurance.");
end
```

Inference - Advantages

Advantages of Inference in Drools

- **Dynamic Decision Making:** Enables the rule engine to make decisions dynamically based on the current state of the working memory.
- **Simplification of Complex Logic:** Breaks down complex decision-making processes into simpler, manageable rules that can work together to deduce conclusions.
- **Flexibility and Scalability:** Easily accommodate new rules and facts, allowing the system to grow and adapt over time.
- **Reduction of Hardcoding Logic:** Reduces the need to hardcode every possible scenario, as the rule engine can infer outcomes based on the defined rules.

Case Study – Lesson 01: Key Takeaways

1. Stateless Sessions are often used for quick, once-off validation or calculation scenarios where there is no need to maintain state or track changes made within session

2. It is good practice to split rules with multiple conditions into smaller, single rules, which are easier to edit or remove while maintaining overall state.

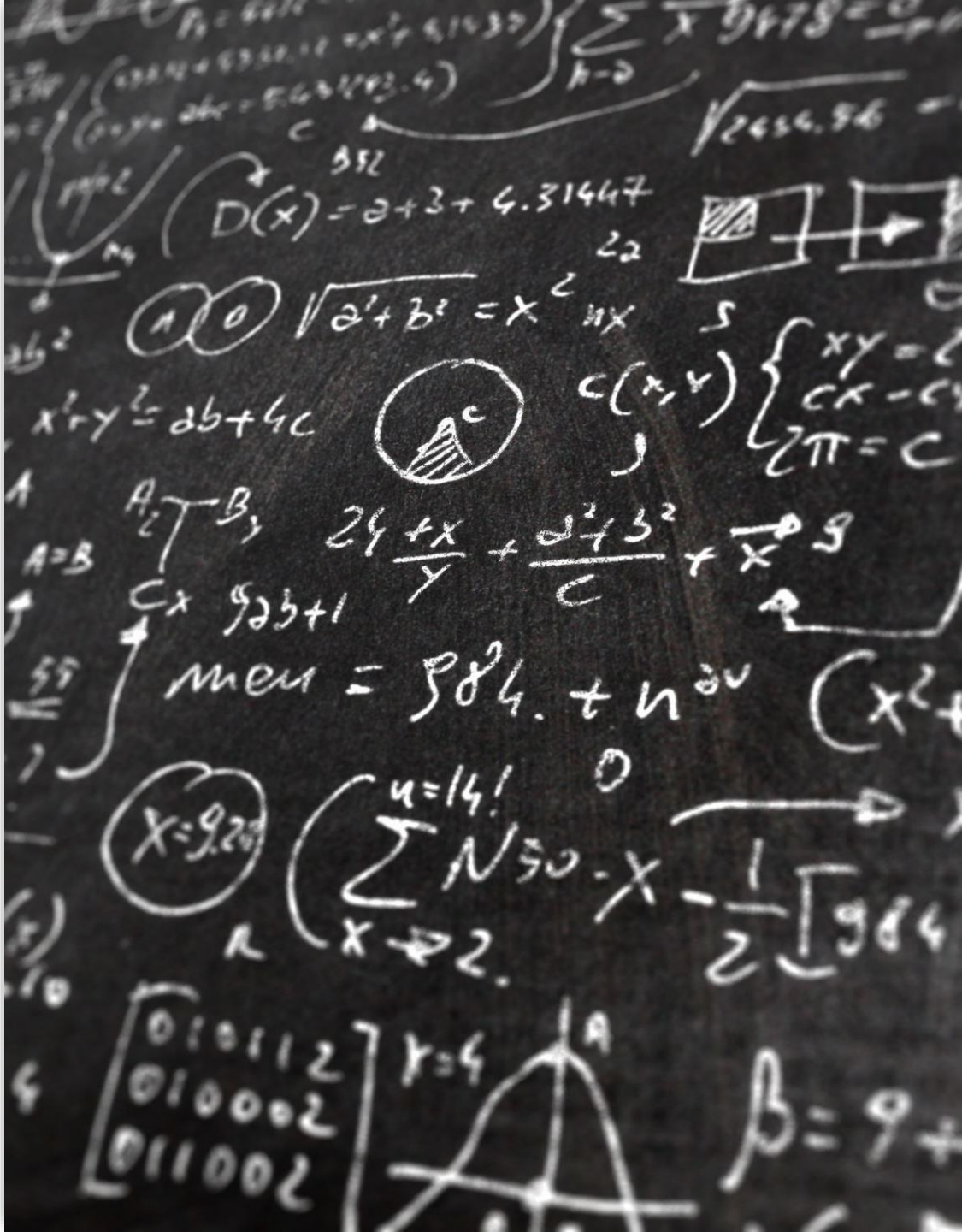
3. Variables can be bound within rules to save objects or object attributes for reporting or attribute modification

4. 'Inference' is a Rule Engine's ability to infer new information based on existing facts without explicit programming.

5. Stateless Sessions are inadequate for maintaining reliable inference – Stateful Sessions are needed for this

Case Study – Lesson 02: Stateful Sessions

Stateful sessions remember the facts that they have processed. They maintain the state of these facts throughout the session's lifecycle. Think of it like a whiteboard where you write information, and this information stays there for reference until you erase it.



Here are some common use-cases for stateful KIE (Knowledge Is Everything) sessions in Drools:

- **Workflow Management**: Managing long-running business processes where state persistence and decision-making evolve over time, like in loan approval processes.
- **Complex Event Processing (CEP)**: Suitable for scenarios where events are correlated and processed in real-time, such as fraud detection in financial transactions.
- **Dynamic Rule-Based Systems**: Systems that require the flexibility to modify rules or facts dynamically, such as adaptive recommendation engines.
- **Real-Time Monitoring**: Ideal for applications that continuously monitor and react to streaming data, like in a security system or stock market analysis.

Aspect	Stateful KIE Sessions	Stateless KIE Sessions
Memory	Session retains memory of processed facts	Does not retain any memory
Fact Modification	Facts can be added, removed, or modified during the session.	Processes a batch of facts without interaction
State Awareness	Aware of the state changes over time and adapts rule processing accordingly.	Not aware of state changes; treats each session independently.
Use Cases	Ideal for complex scenarios where outcomes depend on evolving data	Suitable for lightweight, one-off rule processing scenarios
Resource Utilization	requires more resources to maintain state and respond to changes	more resource-efficient due to low maintenance
Rule Re-Evaluation	Capable of re-evaluating and firing rules as facts are modified	Rules are evaluated once per session without regard for changes

The 'modify' keyword

The 'modify' keyword is used to inform the Drools engine that a fact has been altered. When you modify a fact in a stateful session, Drools re-evaluates the rules to see if this change triggers any new rule or changes the outcome of already fired rules. BEWARE! This can lead to infinite loops.

New Base Object – Insurance Claims

This object represents Insurance claims that have been made by existing clients (we assume) of our insurance company. These claims contain information, such as the insurance Policy Number, the Client's name, and the date of the incident the claim for which the claim is being made. Here is a table to help us visualize the data

Client Name	Policy Number	Date of Claim	Claim Amount
John Doe	P-1100	2022-05-19	8,000
William Johnson	P-1002	2020-04-14	10,000
Michael Davis	P-1004	2023-01-31	40,000
Sophia Martinez	P-1005	2022-01-03	50,000
Ava Thompson	P-1009	2022-02-08	25,000

Lab Exercise

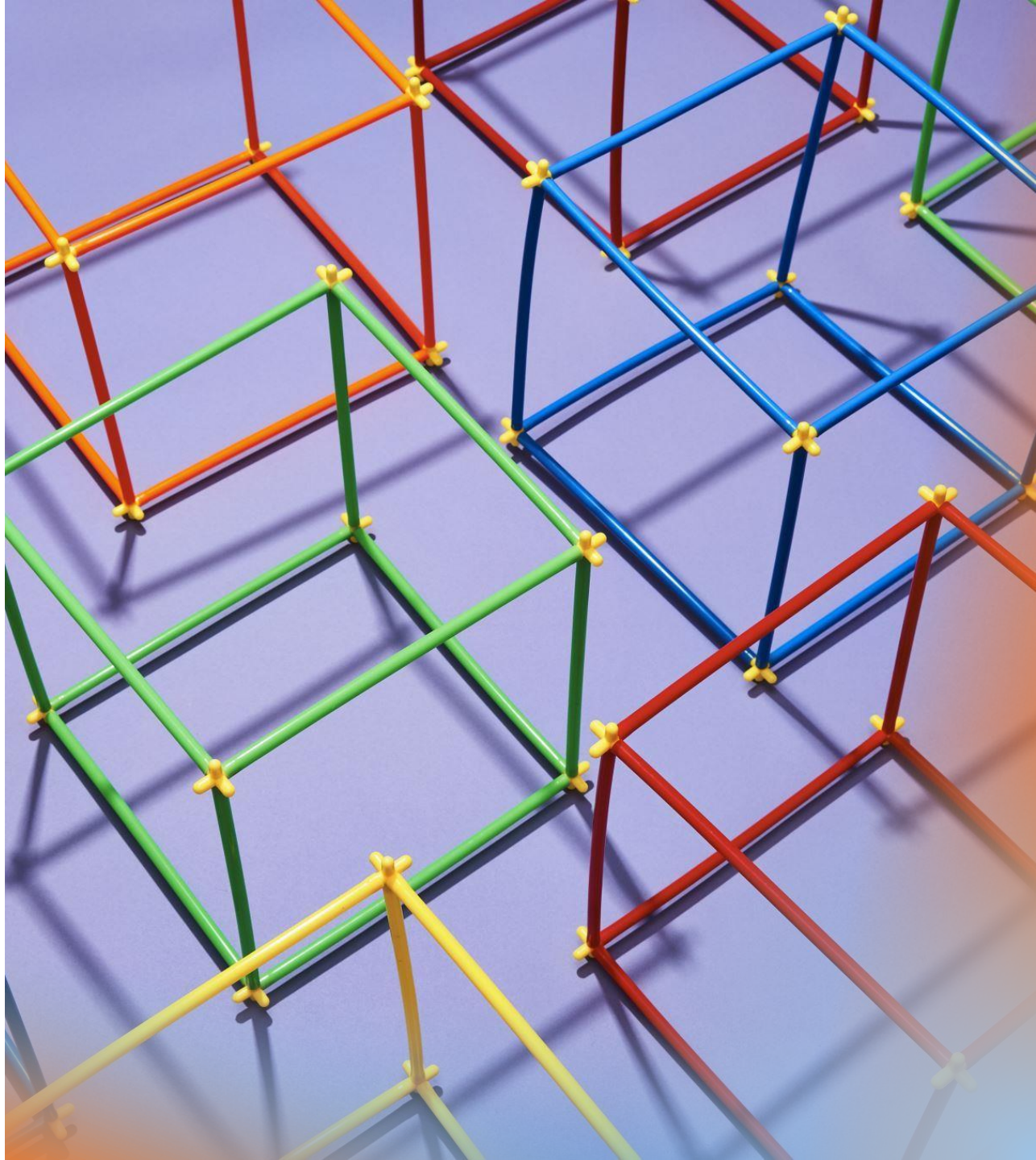
Open and Complete Lab Exercise 02.01 -
Rule Creation for Claims and Policy Management

Case Study – Lesson 02: Key Takeaways

1. Stateful sessions remember the facts that they have processed. They maintain the state of these facts throughout the session's lifecycle.
2. Stateful sessions are the KIE default and do not need to be explicitly declared as do stateless sessions
3. The `modify` keyword makes the session aware of changes made during runtime.
4. Stateless Sessions are needed for reliable inference
5. Stateless Sessions must be explicitly disposed of to prevent excess resource usage and memory leaks

Case Study – Lesson 03: Cross-Products

Cross-Products in Drools refer to the ability to compare or combine data from different fact types (objects) within the rule engine. This capability is crucial for scenarios where decisions depend on relationships between different sets of data.



Cross-Products vs SQL Join

Cross-Products is the Rule Engine equivalent of how the SQL join operation works with tables in a relational database

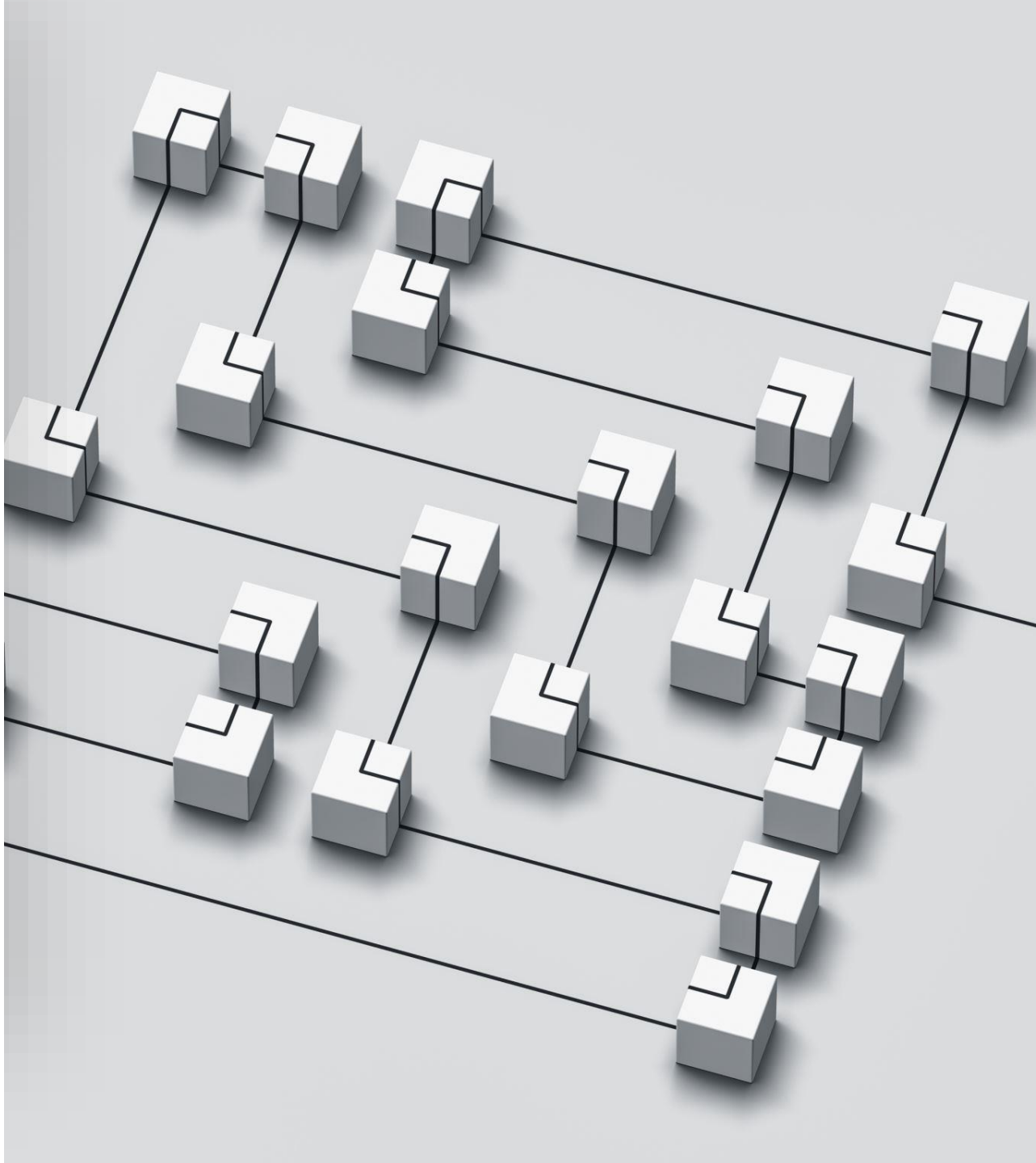
Imagine you're working with two tables in a database: **`Policies`** and **`Claims`**. In SQL, if you want to find claims that relate to specific policies, you might use a join query like this:

```
SELECT Claims.*, Policies.*

FROM Claims

JOIN Policies ON Claims.policyNumber =
    Policies.policyNumber

WHERE Policies.paymentStatus = 'Unpaid';
```



Cross-Products in Drools

In Drools, a similar concept applies when you need to evaluate rules based on information from multiple fact types. For instance, if you have **Claim** and **Policy** objects and you need to assess whether a claim should be approved based on the policy's details (like whether premiums are paid), you would use Drools rules to "join" these facts.

In this Drools example:

The rule "Invalidate claim with unpaid premiums" checks for any **Claim** and associated **Policy** facts where the **policyNumber** fields match and the **Policy**'s **paymentStatus** is "Unpaid".

```
rule "Invalidate claim with unpaid premiums"
    when
        $claim : Claim( ) $policy : Policy( policyNumber
==                $claim.policyNumber, paymentStatus ==
"Unpaid" )
    then
        $claim.setStatus(Status.DENIED);
        System.out.println("Claim denied due to unpaid
premiums for policy " + $policy.policyNumber);
end
```

New Base Object - Policies

This Lesson introduces a new Java Class - `Policy.java` - that represents existing client policies in our system. It contains attributes such as Policy Number, Client Name, Premium, Maximum Coverage, and Payment Status. All Claims must be validated against existing policies. Here is the data on our current clients.

Policy Number	Client Name	Premium	Coverage	Payment Status
P-1000	John Doe	140	14,000	Paid
P-1001	Emma Smith	220	22,000	Unpaid
P-1002	William Johnson	150	15,000	Unpaid
P-1003	Olivia Brown	90	9,000	Paid
P-1004	Michael Davis	400	40,000	Paid
P-1005	Sophia Martinez	230	23,000	Paid
P-1006	Ethan Wilson	180	18,000	Paid
P-1007	Isabella Taylor	240	24,000	Paid
P-1008	Daniel Moore	120	12,000	Unpaid
P-1009	Ava Thompson	210	21,000	Unpaid

New Base Object – Client ID

We now have a second Class that is needed to Validate an Application – a Client ID. Any application must pass a check for validity of both the application and its accompanying ID. This is a simple object with 3 crucial attributes: Name, ID Number, and Expiry Date. Here are all the IDs submitted by prospective clients

Name	ID Number	Expiry Date
James Smith	ID-100	2030-05-21
Maria Smith	ID-101	2023-05-11
Robert Johnson	ID-102	2037-11-11
Linda Johnson	ID-103	2026-11-02
Michael Johnson	ID-104	2032-01-11
Elizabeth Anderson	ID-105	2024-03-31
David Lee	ID-106	2025-06-08
Emily Taylor	ID-107	2027-03-31
William Brown	ID-108	2031-01-08
Jessica Thompson	ID-109	2031-01-08

Lab Exercise

Open and Complete Lab Exercise 02.02 -
Cross-Product Rule Challenge

Case Study – Lesson 03: Key Takeaways

1. Cross-Products is an essential concept when rules are required to evaluate multiple objects
2. Cross-Products is similar in concept to how the SQL join operation is used to access data from multiple tables
3. Separating rules into separate files is good practice for rule organization
4. Cross-Products can lead to unnecessary rule repetition. Use comparison operations to eliminate repeat evaluations.

DROOLS 8

MODULE 05:
FACT OBJECTS, EXECUTION CONTROL, AND TRUTH MAINTENANCE

CONTROLLING RULE FIRING

- **Presented by John Paul Franke**

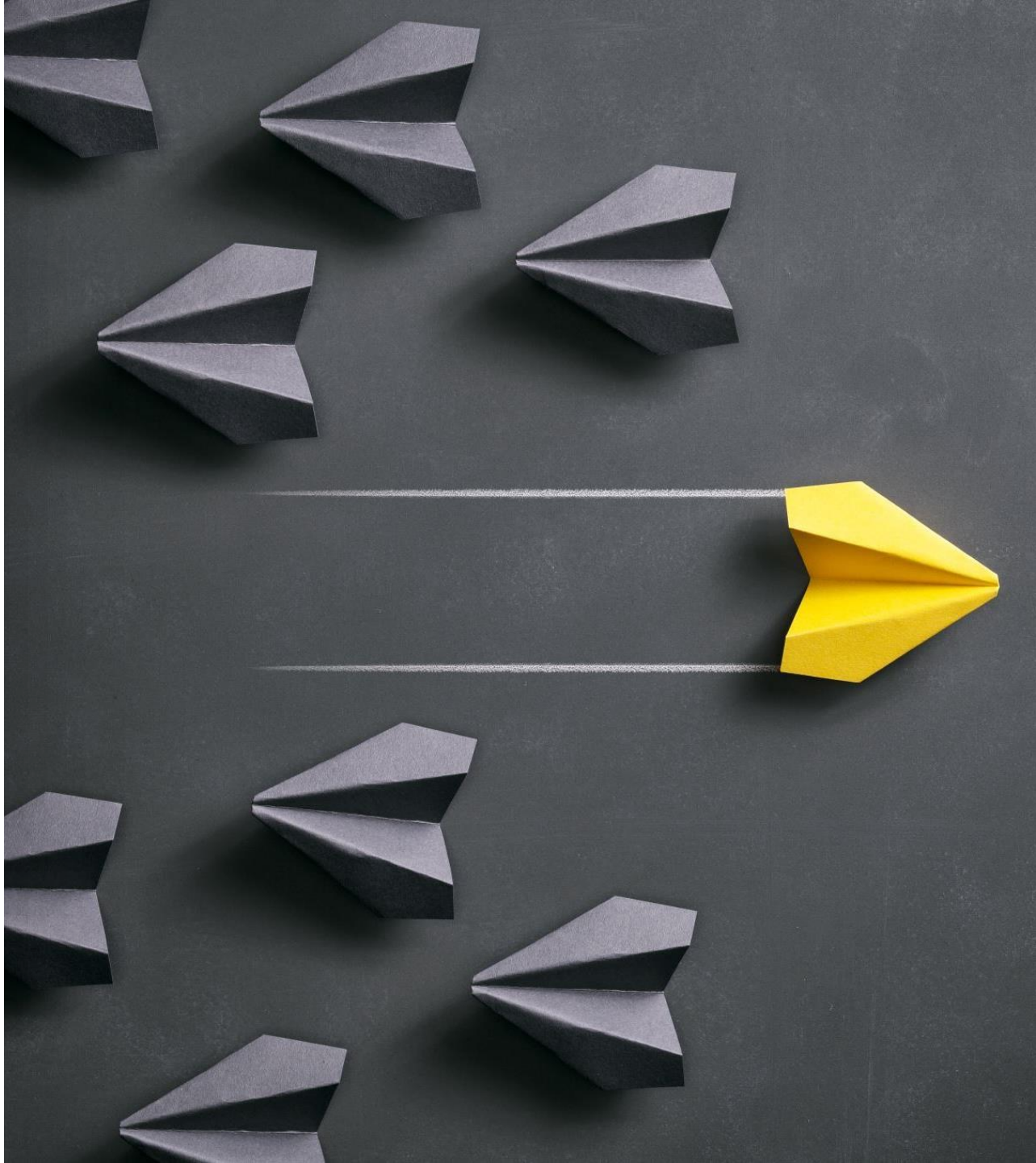
Learning Objectives

Here are our learning goals for this Module:

- Understand object insertion and the use of the **insert** keyword.
- Learn about execution control using salience and agenda groups.
- Explore hybrid control mechanisms combining agenda groups with salience.
- Introduction to activation groups for selective rule firing.
- Understand logical fact insertion and its role in truth maintenance. Deep dive into using fact objects for state management and data separation.
- Implement non-existence checks to prevent redundant rule firing.
- Simplify rules by using implicit validation through logical inserts.

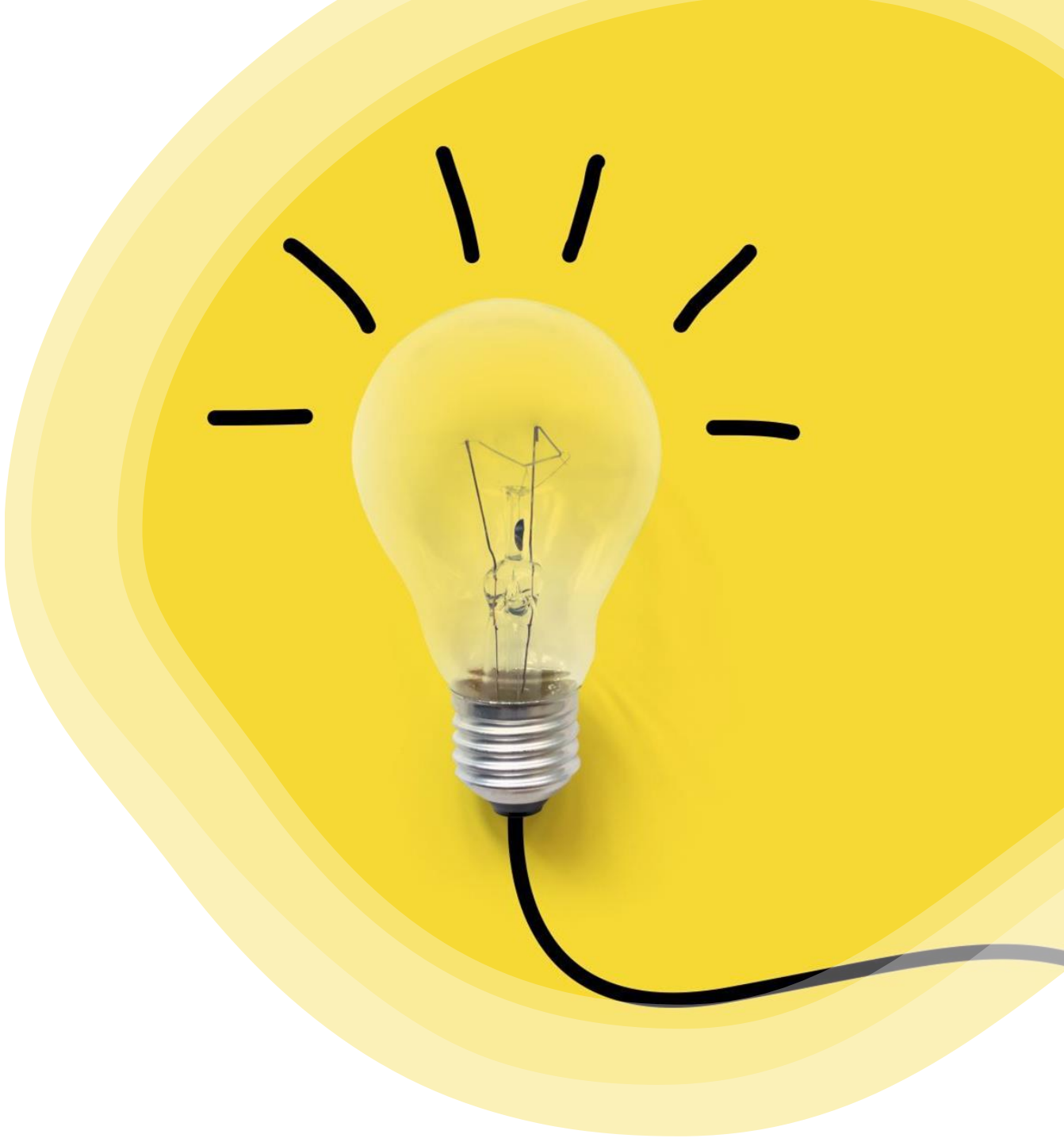
Case Study – Lesson 04: Execution Control & Derived Facts

Execution Controls are the mechanisms by which the Rule Engine can decide which rules to fire, when to fire them, and in what order.



Derived Facts

A **"derived fact"** represents a piece of data that is not part of the original input provided to the rule engine but is generated during the rule engine's execution based on the logic defined within the rules. Derived facts are created by rules being fired and can themselves trigger other rules, enabling complex decision-making processes and allowing the knowledge session to evolve dynamically.



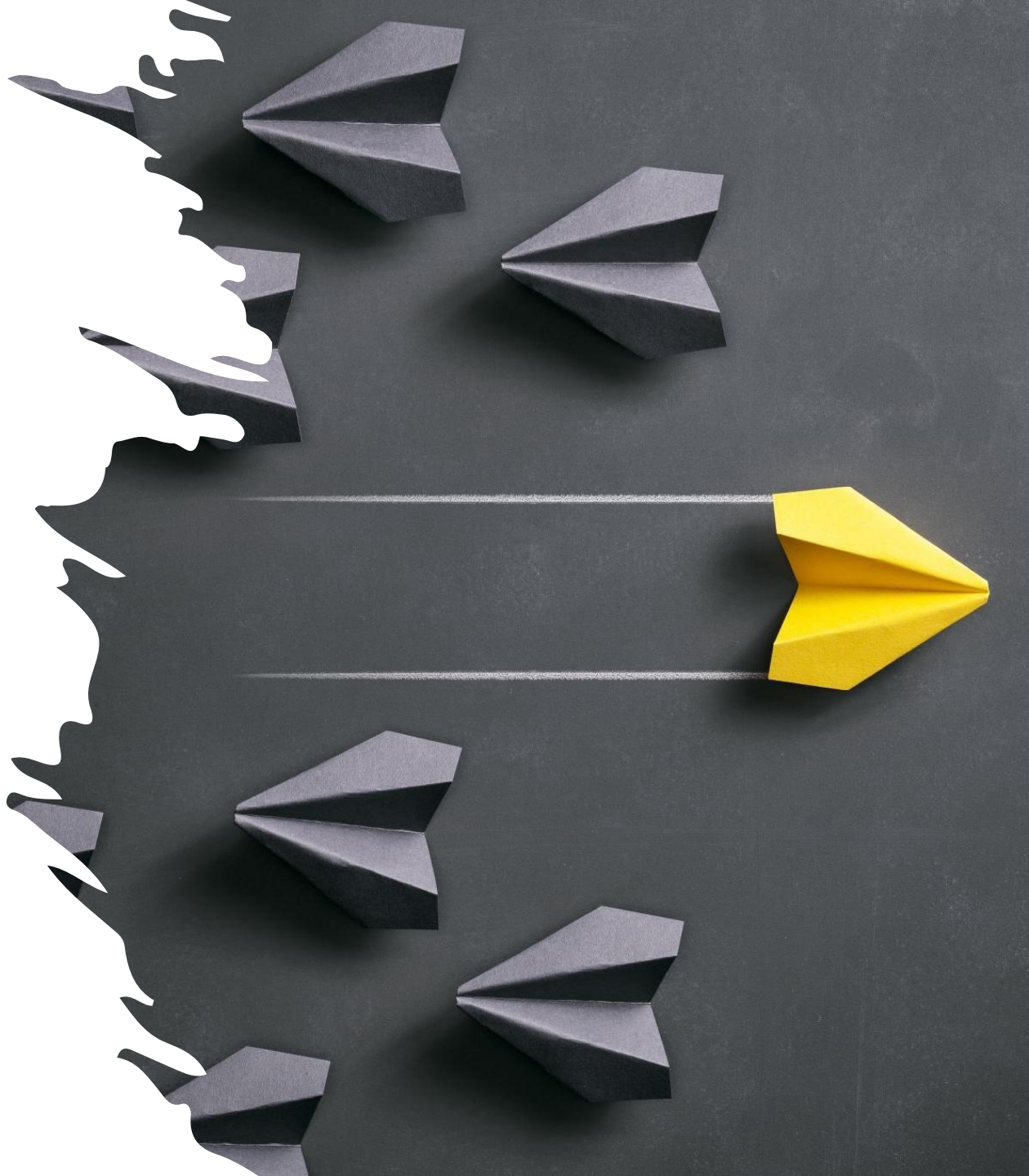
The `insert` keyword

The `insert` keyword allows us to insert derived facts into a drools session. When a rule's conditions (the "when" part) are met, the actions specified in the "then" part are executed. If the **`insert`** keyword is used in the "then" part, it will create a new fact object and add it to the working memory.

Note: The use of **`insert`** can lead to a significant increase in the number of facts within the working memory, influencing memory usage and performance.

Execution Control - Salience

The 'salience' attribute assigns a priority – represented as a numerical value – to rules. Rules with higher salience values are executed before those with lower values. By default, all rules have a salience of 0. Salience can be used to control the order explicitly.



Lab Exercise

Open and Complete Lab Exercise 02.03 -
Rule Execution Control with Salience

Execution Control - Agenda Groups

Agenda groups allow grouping of rules into sets that can be executed as a batch. Only rules in the agenda group that has focus are considered for execution. This can control the execution flow by activating or deactivating groups.

Agenda Group Stack

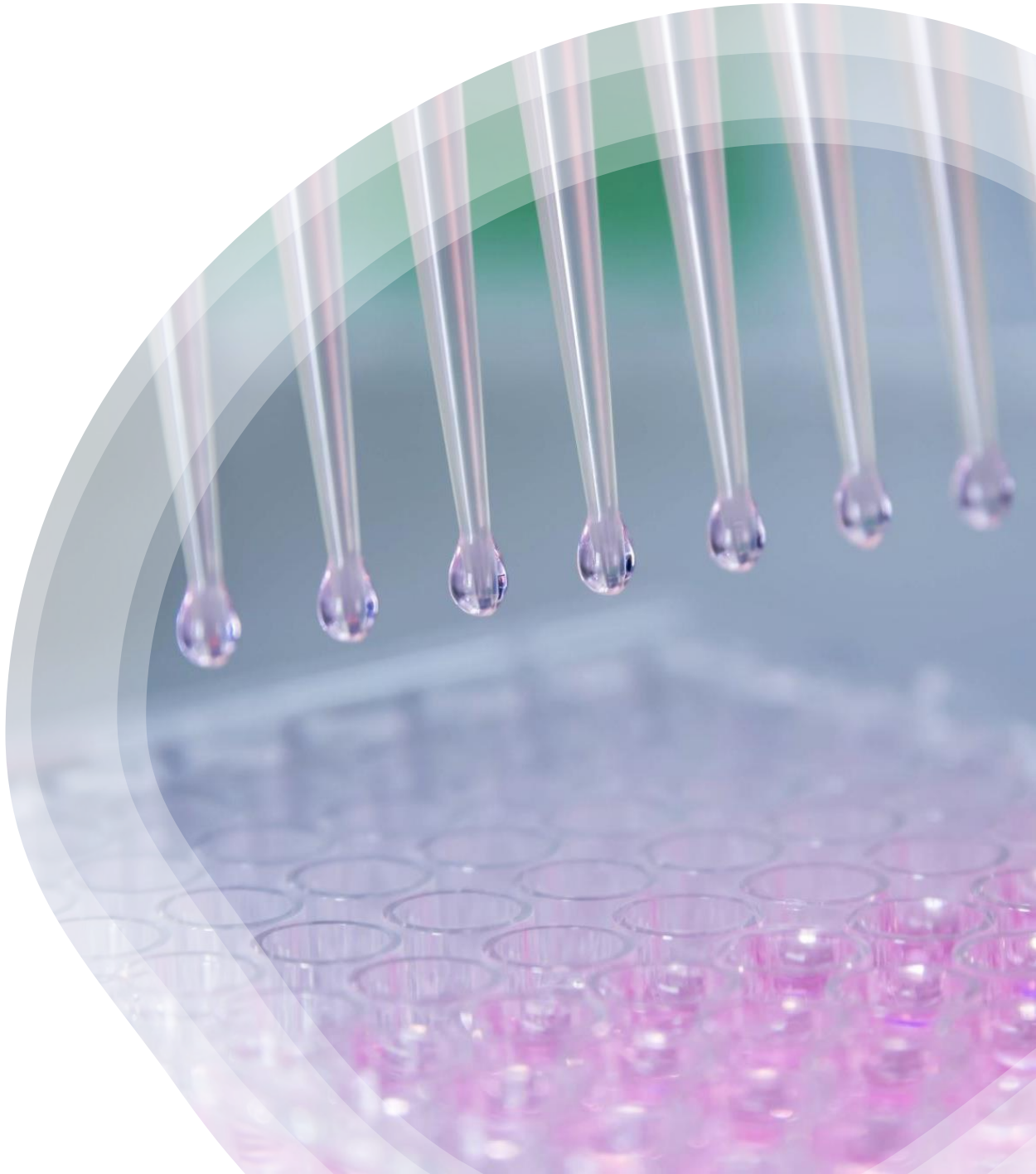
With agenda groups, a 'stack' is a mechanism for managing the execution order of rules grouped within different agenda groups.

A stack uses 'first in, last out' logic to build and fire the stack's agenda groups.

How it Works

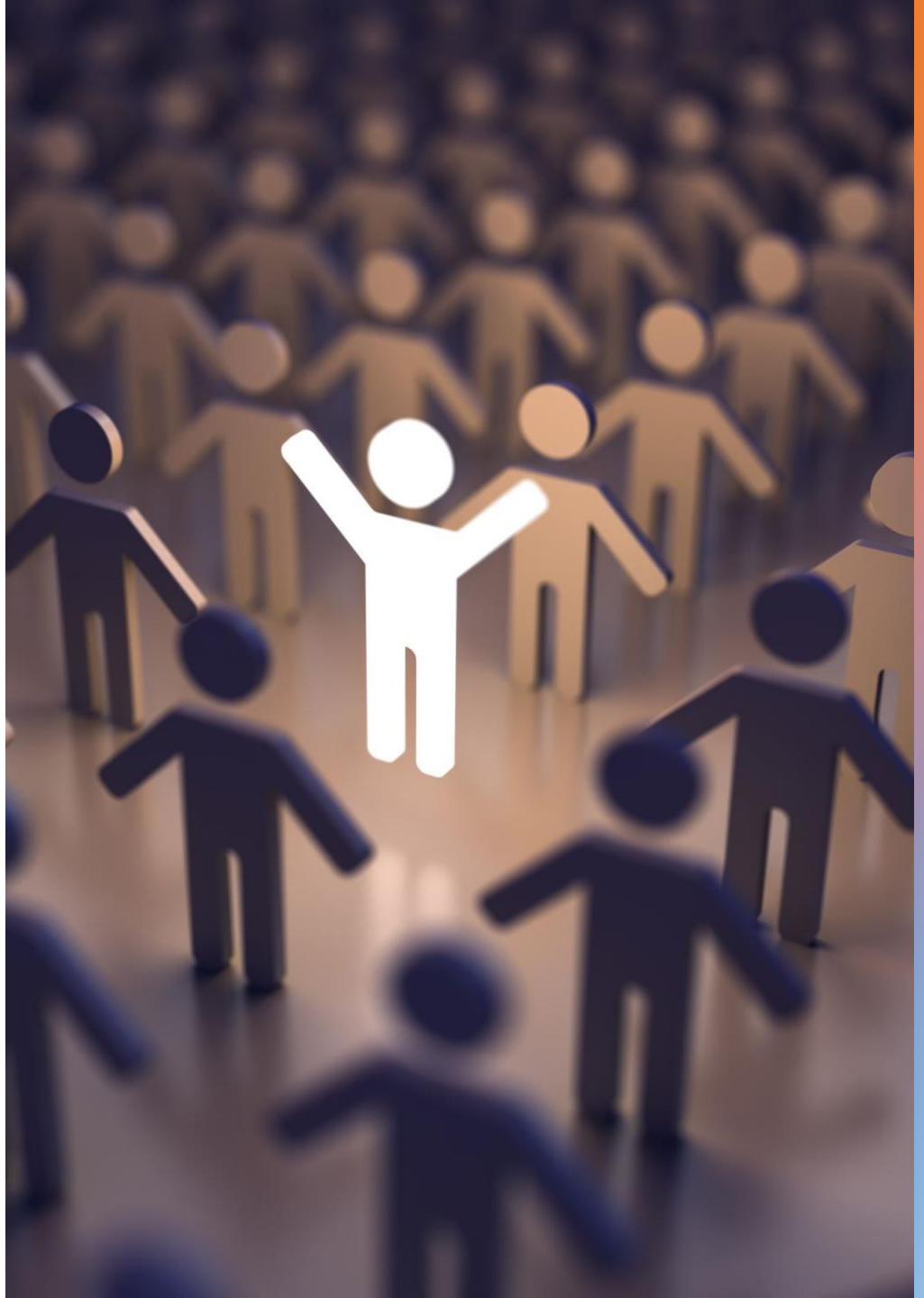
Open and Complete Lab Exercise 02.04 - **Rule Execution Control with Agenda Groups**

Lab Exercise



Execution Control - Activation Groups

The 'activation-group' attribute ensures that only one rule within the same activation group fires. If multiple rules in an activation group become eligible for execution, only the one with the highest priority (salience) will be executed.



Lab Exercise

Open and Complete Lab Exercise 02.05 -
Rule Modification Challenge

Case Study – Lesson 04: Key Takeaways

1. The insertion of 'derived facts' into a running session can insert additional data into the working memory, enabling the rule engine to adapt its processing, potentially influencing the agenda and the firing of rules.
2. The `insert` keyword dynamically inserts new derived facts into a running session.
3. Salience is used to explicitly control the order of rule firing based on numerical values
4. Agenda-groups create rule groups that can be fired in order using a 'stack'
5. Salience can be combined with agenda groups to control rule firing order within groups
6. Activation-groups will select only a single rule – the one with the highest salience – to fire within a group.

Truth maintenance in Drools, also known as logical assertion or logical insertion, is a feature that helps manage the logical implications of facts within a Drools rule engine session.

As a Detective, you gather clues to solve a mystery. As you find new clues, your theories about the mystery might change. If a clue that led to a theory is proven wrong, you'd naturally discard the theory. Truth maintenance in Drools works similarly with facts and rules in a knowledge session.

Case Study – Lesson 05: Truth Maintenance and Logical Facts

Logical Fact Insertion

We use the 'insertLogical' keyword to perform truth maintenance. When you assert a fact logically using Drools, you're saying, "This fact is true because the conditions for these other facts are true." When a fact is logically inserted, it's tied to the condition that created it. If that condition is no longer valid, the fact is automatically retracted (deleted) from the session.

Fact Objects vs Attribute modification

Sometimes, instead of modifying the properties of a base object, we will want to use an object wrapper, the same way we might put a 'valid' or 'invalid' stamp or sticker on an application.

Fact objects are often preferred over directly editing the properties of base objects for several reasons:

Immutability and State Management: Using Fact objects helps in maintaining immutability, preventing unwanted side effects and ensuring consistency

Clarity and Maintainability: By using Fact objects, you create a clear separation between the raw data (base objects) and the processed or contextual data (Fact objects)

Performance: Drools is designed to work with fact objects in its working memory. By inserting, updating, or retracting Fact objects instead of modifying base objects directly, The rule engine can more efficiently detect changes and react to them.

Audit Trail and Transparency: When using Fact objects, it’s easier to maintain an audit trail of what changes were made, when, and by which rule.

New Fact Objects

This lesson introduces several new Fact Objects:

Valid Application

Valid Claim

Valid ClientID

Invalid Application

Invalid Claim

Invalid ClientID

These objects will act as a 'stamp' on our base facts.

Case Study – Lesson 05: Key Takeaways

1. Truth Maintenance helps manage the logical implications of facts within a Drools rule engine session.
2. The `insertLogical` keyword inserts facts into a session that are tied to the truth of a rule's Left-Hand Side.
3. Fact Objects are often used as validation stamps on base objects and can help maintain audit trail, performance optimization, and state management
4. Check for the 'non-existence' of fact objects to avoid unnecessary rule repetition.
5. Use 'Implicit Validation' to reduce number of rules

DROOLS 8

MODULE 06: OBJECT GROUPS

DEALING WITH COLLECTIONS

Presented by John Paul Franke

Learning Objectives

Here are our learning goals for this Module:

- Learn to manipulate groups of objects using DRL keywords
  - `memberOf`
  - `exists`
  - `forall`
- Learn to control rule validity with attributes
- Put our knowledge to the test with advanced labs and milestone project

Case Study – Lesson 06: Object Groups

In Drools, rules can interact with groups or collections of objects and attributes through several mechanisms, allowing for complex decision-making processes based on collections of data.

New Base Object – Family Application

We now allow for family visa applications with several applicants. Here is what our applications look like as a data table:

Application Number	Client ID Numbers	Client Ages	Start Date	Policy Duration Years
GIA-1000	ID-100 ID-101	52, 56	2024-03-01	1
GIA-1001	ID-102 ID-103 ID-104	38, 36, 16	2024-05-05	1

Case Study – Lesson 06: Object Groups

In Drools, rules can interact with groups or collections of objects and attributes through several mechanisms, allowing for complex decision-making processes based on collections of data.

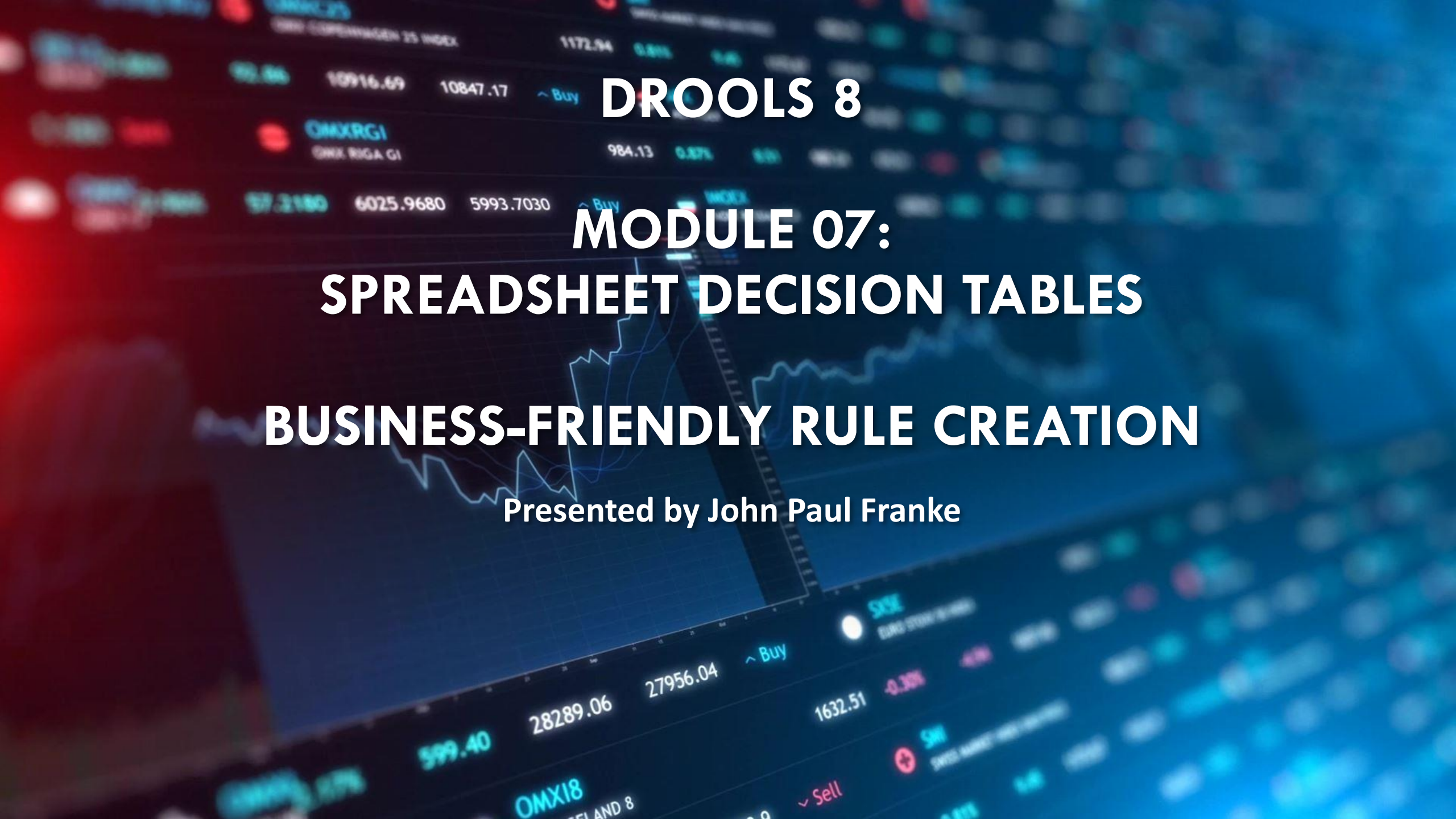
Case Study – Lesson 06: Key Takeaways

1. Drools uses several mechanisms for dealing with collections in rules
2. The 'memberOf' keyword is used to check if an element is a 'member of' a collection or array
3. The 'exists' keyword performs a single check for the existence of a fact that meets a specified criteria and can help avoid unnecessary reevaluations
4. The 'forall' keyword is used to evaluate that a condition is true 'for all' objects of a collection or array

Lab Exercise

Open and Complete Lab Exercise 02.06 -
Dynamic Premium Calculation Based on Applicant Attributes

Thank You!

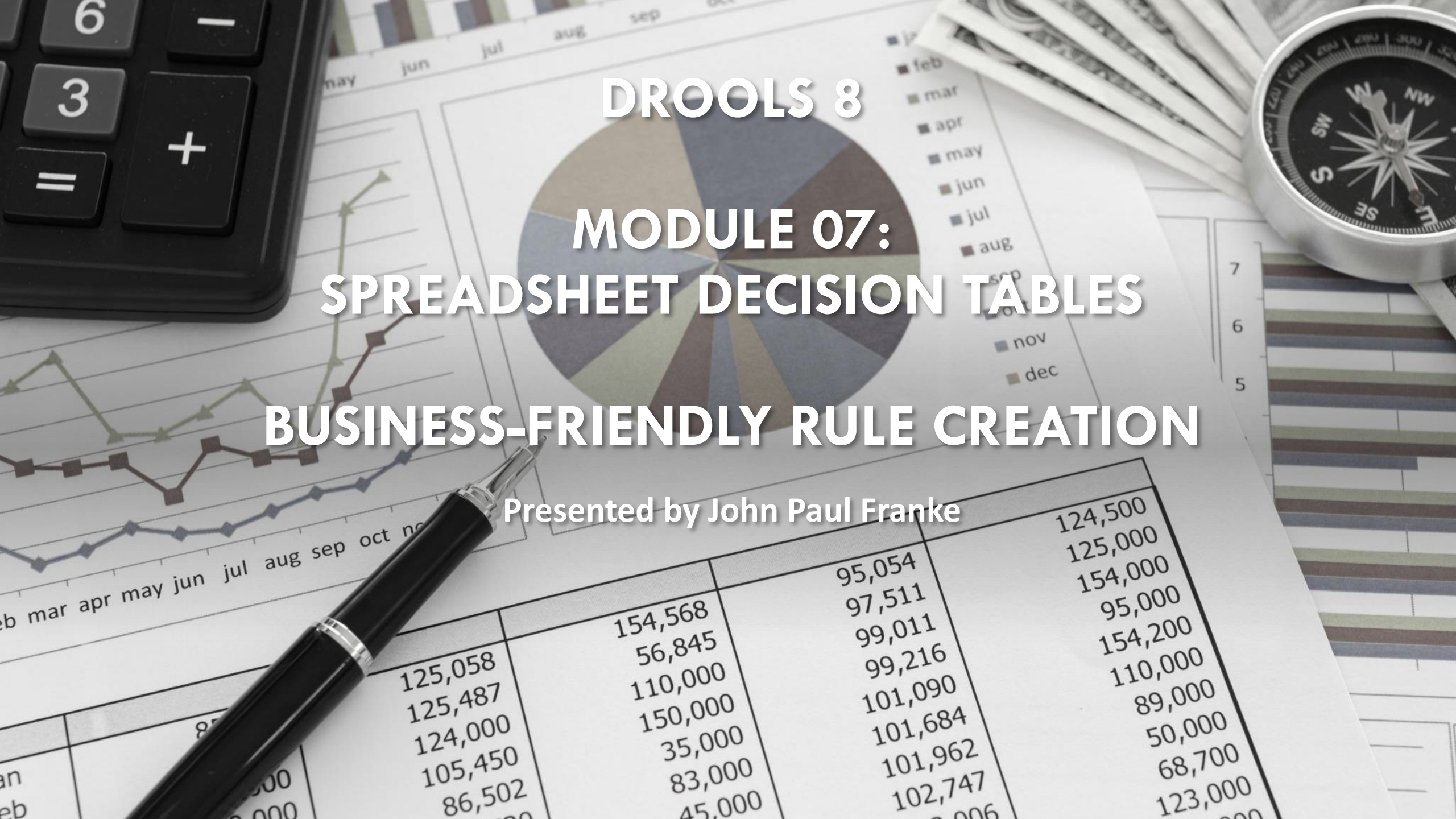


DROOLS 8

MODULE 07: SPREADSHEET DECISION TABLES

BUSINESS-FRIENDLY RULE CREATION

Presented by John Paul Franke



DROOLS 8

MODULE 07: SPREADSHEET DECISION TABLES

BUSINESS-FRIENDLY RULE CREATION

Presented by John Paul Franke

Learning Objectives

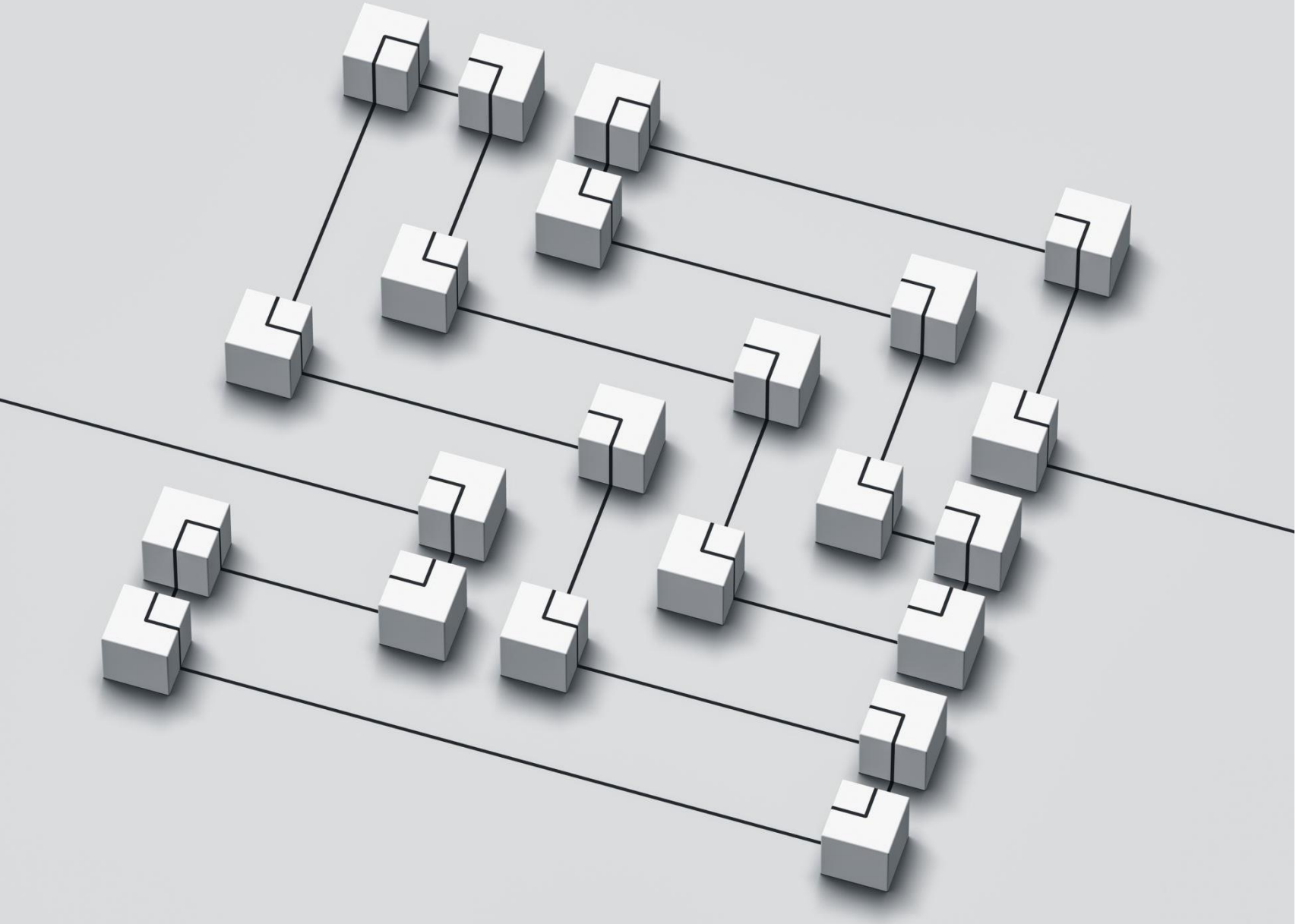
Here are our learning goals for this Module:

- Understand How Spreadsheet Decision Tables Help bridge the gap between non-technical business analysts and technical developers
- Learn how SDT's are created
- Practice the logic and syntax of creating rules in a SDT

Spreadsheet Decision Tables

Spreadsheet decision tables allow for the definition of business rules in a tabular format that is familiar and accessible to non-technical users. They serve as a bridge between technical and business stakeholders by presenting rules in a structured and clear manner.

The development of decision tables involves close collaboration between business analysts and developers – business analysts define the rules in business language, which are then mapped to technical scripts by developers.



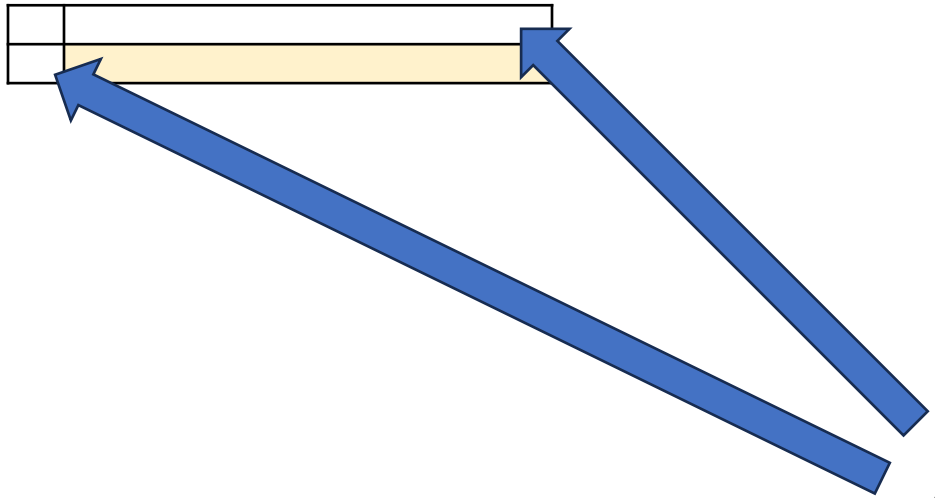
Structure and Syntax

A decision table is composed of a **`RuleSet`** area that defines global settings and **`RuleTable`** areas that contain the actual rule definitions.

Integration with Drools

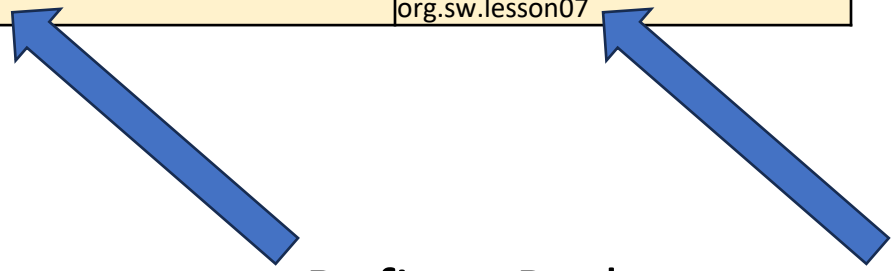
Decision tables are compiled into Drools Rule Language (DRL) when processed by the Drools engine.

They must be properly formatted and follow the syntax requirements to be correctly compiled.



Optional Blank Rows/Collumns allow for notes, comments, descriptive metadata, etc (all will be ignored)

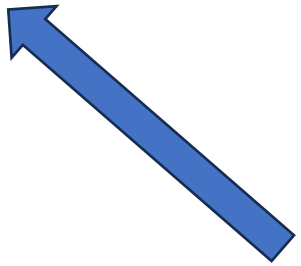
	RuleSet	org.sw.lesson07



Defines Package and Marks start of Decision Table

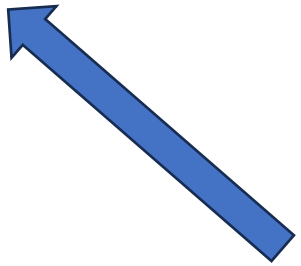
Package Name

	RuleSet	org.sw.lesson07
	Notes	This Section demonstrates various features of how to create Spreadsheet Decision Tables



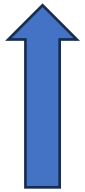
Personal Notes
(Optional)

	RuleSet	org.sw.lesson07
	Notes	This Section demonstrates various features of how to create Spreadsheet Decision Tables
	import	org.sw.facts.Application



Object Imports

	RuleSet	org.sw.lesson07
	Notes	This Section demonstrates various features of how to create Spreadsheet Decision Tables
	import	org.sw.facts.Application, org.sw.facts.Proposal



Multiple imports defined in same cell, separated by comma

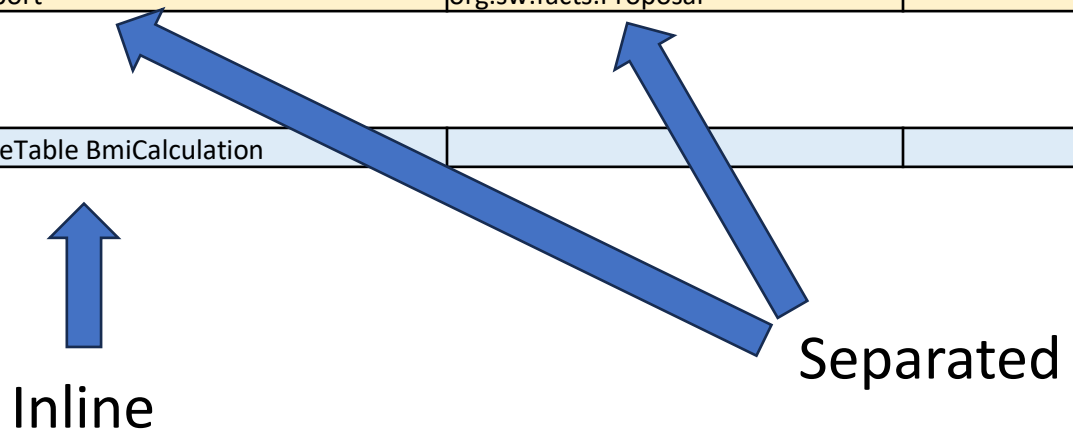
	RuleSet	org.sw.lesson07	
	Notes	This Section demonstrates various features of how to create Spreadsheet Decision Tables	
	import	org.sw.facts.Application, org.sw.facts.Proposal	

	RuleTable BmiCalculation		

Mandatory Whitespace before RuleTable

Defines a RuleTable (like a .drl file)

	RuleSet	org.sw.lesson07	
	Notes	This Section demonstrates various features of how to create Spreadsheet Decision Tables	
	import	org.sw.facts.Application, org.sw.facts.Proposal	
	RuleTable BmiCalculation		



	RuleSet	org.sw.lesson07	
	Notes	This Section demonstrates various features of how to create Spreadsheet Decision Tables	
	import	org.sw.facts.Application, org.sw.facts.Proposal	

	RuleTable BmiCalculation		
	CONDITION	ACTION	ACTION

LHS: Independent Variables

RHS: Dependent Variables

	RuleSet	org.sw.lesson07	
	Notes	This Section demonstrates various features of how to create Spreadsheet Decision Tables	
	import	org.sw.facts.Application, org.sw.facts.Proposal	

	RuleTable BmiCalculation		
	CONDITION	ACTION	ACTION
	$application: Application()		

Fact Object to be evaluated

	RuleSet	org.sw.lesson07	
	Notes	This Section demonstrates various features of how to create Spreadsheet Decision Tables	
	import	org.sw.facts.Application, org.sw.facts.Proposal	

	RuleTable BmiCalculation		
	CONDITION	CONDITION	ACTION
	$application: Application()		

Also Possible: Single fact object used across multiple conditions

	RuleSet	org.sw.lesson07	
	Notes	This Section demonstrates various features of how to create Spreadsheet Decision Tables	
	import	org.sw.facts.Application, org.sw.facts.Proposal	

	RuleTable BmiCalculation		
	CONDITION	CONDITION	ACTION
	$application: Application()	$clientId: clientID()	

Also Possible: multiple fact objects across multiple conditions (Allows Cross-Product Evaluations)

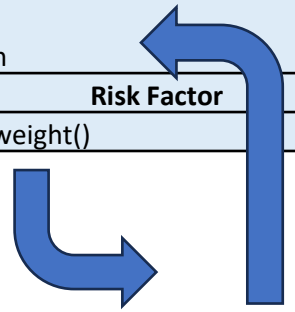
	RuleSet	org.sw.lesson07		
	Notes	This Section demonstrates various features of how to create Spreadsheet Decision Tables		
	import	org.sw.facts.Application, org.sw.facts.Proposal		

	RuleTable BmiCalculation			
	CONDITION	CONDITION		ACTION
	$application: Application()	$clientId: clientID()		
		IdNumber == $param	expired	System.*out*.println($application + " Invalid – ID number" + $param + " expired");
	Exists	**Items Match**	**Expiry Status**	**ID Number**
	TRUE	$application.clientIdNumber	TRUE	$clientId.idNumber

Also Possible: multiple fact objects across multiple conditions (Allows Cross-Product Evaluations)

	RuleSet	org.sw.lesson07	
	Notes	This Section demonstrates various features of how to create Spreadsheet Decision Tables	
	import	org.sw.facts.Application, org.sw.facts.Proposal	

	RuleTable BmiCalculation		
	CONDITION	ACTION	ACTION
	$application: Application()		
	$param	System.out.println("Positive Risk Factor in application "+$application.getApplicationNumber() +". Premium adjustment: "+$param);	$application.addPremium($param);
	Risk Factor	**Premium Adjustment**	
	isOverweight()	10	



Output of method contained in `$param` variable

Actions taken on fact object, using variable

This row is the actual Rule

	RuleSet	org.sw.lesson07	
	Notes	This Section demonstrates various features of how to create Spreadsheet Decision Tables	
	import	org.sw.facts.Application, org.sw.facts.Proposal	

	RuleTable BmiCalculation		
	CONDITION	ACTION	ACTION
	$application: Application()		
	Overweight	System.out.println("Positive Risk Factor in application "+$application.getApplicationNumber() +". Premium adjustment: "+$param);	$application.addPremium($param);
	Value	**Premium Adjustment**	
	TRUE	10	

Actions taken on fact object, using variable

Also Possible: Attribute or method accessed directly using logical name and compared to value

	RuleSet	org.sw.lesson07	
	Notes	This Section demonstrates various features of how to create Spreadsheet Decision Tables	
	import	org.sw.facts.Application, org.sw.facts.Proposal	

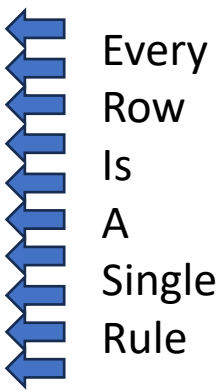
	RuleTable BmiCalculation		
	CONDITION	ACTION	ACTION
	$application: Application()		
	$param	System.out.println("Positive Risk Factor in application "+$application.getApplicationNumber() +". Premium adjustment: "+$param);	$application.addPremium($param);
	Risk Factor	**Premium Adjustment**	
	isOverweight()	10	

FINAL RULE

```
// rule values at B13, header at B8
rule "PremiumCalculation_13"
   when
      $application: Application(isOverweight())
   then
      System.out.println( "Positive Risk Factor in application " +$application.getApplicationNumber() +". Premium adjustment: " + 10);
      $application.addPremium( 10 );
end
```

	RuleSet	org.sw.lesson07	
	Notes	This Section demonstrates various features of how to create Spreadsheet Decision Tables	
	import	org.sw.facts.Application, org.sw.facts.Proposal	

	RuleTable BmiCalculation		
	CONDITION	ACTION	ACTION
	$application: Application()		
	$param	System.out.println("Positive Risk Factor in application "+$application.getApplicationNumber() +". Premium adjustment: "+$param);	$application.addPremium($param);
	Risk Factor	**Premium Adjustment**	
	isOverweight()	10	
	isObeseA()	20	
	isObeseB()	30	
	isObeseC()	40	
	isSmoker()	15	
	risk == "high"	30	
	risk == "medium"	15	
	clientAge >= 70	20	
	PEC	20	



	RuleSet	org.sw.lesson07	
	Notes	This Section demonstrates various features of how to create Spreadsheet Decision Tables	
	import	org.sw.facts.Application, org.sw.facts.Proposal	

Marker for Rule Comments

	RuleTable BmiCalculation			
	CONDITION	ACTION	ACTION	DESCRIPTION
	$application: Application()			
	$param	System.out.println("Positive Risk Factor in application "+$application.getApplicationNumber() +". Premium adjustment: "+$param);	$application.addPremium($param);	
	Risk Factor	**Premium Adjustment**		**Comments**
	isOverweight()	10		10% increase for overweight BMI
	isObeseA()	20		20% increase for overweight BMI
	isObeseB()	30		30% increase for overweight BMI
	isObeseC()	40		40% increase for overweight BMI
	isSmoker()	15		15% increase for overweight BMI
	risk == "high"	30		30% increase for overweight BMI
	risk == "medium"	15		15% increase for overweight BMI
	clientAge >= 70	20		20% increase for overweight BMI
	PEC	20		20% increase for overweight BMI

	RuleSet	org.sw.lesson07	
	Notes	This Section demonstrates various features of how to create Spreadsheet Decision Tables	
	import	org.sw.facts.Application, org.sw.facts.Proposal	

Declare name for rules

	RuleTable BmiCalculation				
	CONDITION	ACTION	ACTION	DESCRIPTION	NAME
	$application: Application()				
	$param	System.out.println("Positive Risk Factor in application "+$application.getApplicationNumber() +". Premium adjustment: "+$param);	$application.addPremium($param);		
	Risk Factor	**Premium Adjustment**		**Comments**	**Rule Name**
	isOverweight()	10		10% increase for overweight BMI	Adjust premium for overweight client
	isObeseA()	20		20% increase for overweight BMI	Adjust premium for Obese Class A client
	isObeseB()	30		30% increase for overweight BMI	Adjust premium for Obese Class B client
	isObeseC()	40		40% increase for overweight BMI	Adjust premium for Obese Class C client
	isSmoker()	15		15% increase for overweight BMI	Adjust premium for smokers
	risk == "high"	30		30% increase for overweight BMI	Adjust premium for high-risk client
	risk == "medium"	15		15% increase for overweight BMI	Adjust premium for medium-risk client
	clientAge >= 70	20		20% increase for overweight BMI	Adjust premium for seniors
	PEC	20		20% increase for overweight BMI	Adjust premium for clients with PEC

	RuleSet	org.sw.lesson07	
	Notes	This Section demonstrates various features of how to create Spreadsheet Decision Tables	
	import	org.sw.facts.Application, org.sw.facts.Proposal	

Multiple Rule tables:

- Like having a second .drl file
- Separated by blank row
- Marked by 'RuleTable' cell

	RuleTable BmiCalculation				
	CONDITION	ACTION	ACTION	DESCRIPTION	NAME
	$application: Application()				
	$param	System.out.println("Positive Risk Factor in application "+$application.getApplicationNumber() +". Premium adjustment: "+$param);	$application.addPremium($param);		
	Risk Factor	**Premium Adjustment**		**Comments**	**Rule Name**
	isOverweight()	10		10% increase for overweight BMI	Adjust premium for overweight client
	isObeseA()	20		20% increase for overweight BMI	Adjust premium for Obese Class A client
	isObeseB()	30		30% increase for overweight BMI	Adjust premium for Obese Class B client
	isObeseC()	40		40% increase for overweight BMI	Adjust premium for Obese Class C client
	isSmoker()	15		15% increase for overweight BMI	Adjust premium for smokers
	risk == "high"	30		30% increase for overweight BMI	Adjust premium for high-risk client
	risk == "medium"	15		15% increase for overweight BMI	Adjust premium for medium-risk client
	clientAge >= 70	20		20% increase for overweight BMI	Adjust premium for seniors
	PEC	20		20% increase for overweight BMI	Adjust premium for clients with PEC

	RuleTable IssueProposal				
	CONDITION	ACTION	ACTION	DESCRIPTION	NAME
	$application: Application()				
		System.out.println($1+" Final Premium Calculation: " + $2);	insert(new Proposal($1, $2));		
	Exists	**No Parameters Needed**		**Comments**	**Rule Name**
	TRUE	$application.getApplicationNumber(), $application.getPremium()		Issues proposal after adjustments	Issue insurance proposal

	RuleSet	org.sw.lesson07	
	Notes	This Section demonstrates various features of how to create Spreadsheet Decision Tables	
	import	org.sw.facts.Application, org.sw.facts.Proposal	

`PRIORITY` Denotes Salience

	RuleTable B[illegible]ation				
	PRIORITY	CONDITION	ACTION	ACTION	DESCRIPTION
		$application: Application()			
		$param	System.out.println("Positive Risk Factor in application "+$application.getApplicationNumber() +". Premium adjustment: "+$param);	$application.addPremium($param);	
	Priority	**Risk Factor**	**Premium Adjustment**		**Comments**
	100	isOverweight()	10		10% increase for overweight BMI
	100	isObeseA()	20		20% increase for overweight BMI
	100	isObeseB()	30		30% increase for overweight BMI
	100	isObeseC()	40		40% increase for overweight BMI
	100	isSmoker()	15		15% increase for overweight BMI
	100	risk == "high"	30		30% increase for overweight BMI
	100	risk == "medium"	15		15% increase for overweight BMI
	100	clientAge >= 70	20		20% increase for overweight BMI
	100	PEC	20		20% increase for overweight BMI

	RuleTable IssueProposal	RuleTable IssueProposal			
	PRIORITY	CONDITION	ACTION	ACTION	DESCRIPTION
		$application: Application()			
			System.out.println($1+" Final Premium Calculation: " + $2);	insert(new Proposal($1, $2));	
	Priority	**Exists**	**No Parameters Needed**		**Comments**
	0	TRUE	$application.getApplicationNumber(), $application.getPremium()		Issues proposal after adjustments

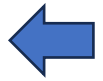
	RuleSet	org.sw.lesson07	
	Notes	This Section demonstrates various features of how to create Spreadsheet Decision Tables	
	import	org.sw.facts.Application, org.sw.facts.Proposal	
	Sequential	TRUE	

Sets Automatic descending salience, starting from 65535

	RuleTable BmiCalculation				
	CONDITION	ACTION	ACTION	DESCRIPTION	NAME
	$application: Application()				
	$param	System.out.println("Positive Risk Factor in application "+$application.getApplicationNumber() +". Premium adjustment: "+$param);	$application.addPremium($param);		
	Risk Factor	**Premium Adjustment**		**Comments**	**Rule Name**
	isOverweight()	10		10% increase for overweight BMI	Adjust premium for overweight client
	isObeseA()	20		20% increase for overweight BMI	Adjust premium for Obese Class A client
	isObeseB()	30		30% increase for overweight BMI	Adjust premium for Obese Class B client
	isObeseC()	40		40% increase for overweight BMI	Adjust premium for Obese Class C client
	isSmoker()	15		15% increase for overweight BMI	Adjust premium for smokers
	risk == "high"	30		30% increase for overweight BMI	Adjust premium for high-risk client
	risk == "medium"	15		15% increase for overweight BMI	Adjust premium for medium-risk client
	clientAge >= 70	20		20% increase for overweight BMI	Adjust premium for seniors
	PEC	20		20% increase for overweight BMI	Adjust premium for clients with PEC

	RuleTable IssueProposal				
	CONDITION	ACTION	ACTION	DESCRIPTION	NAME
	$application: Application()				
		System.out.println($1+" Final Premium Calculation: " + $2);	insert(new Proposal($1, $2));		
	Exists	**No Parameters Needed**		**Comments**	**Rule Name**
	TRUE	$application.getApplicationNumber(), $application.getPremium()		Issues proposal after adjustments	Issue insurance proposal

	RuleSet	org.sw.lesson07	
	Notes	This Section demonstrates various features of how to create Spreadsheet Decision Tables	
	import	org.sw.facts.Application, org.sw.facts.Proposal	
	Sequential	TRUE	
	SequentialMaxPriority	100	



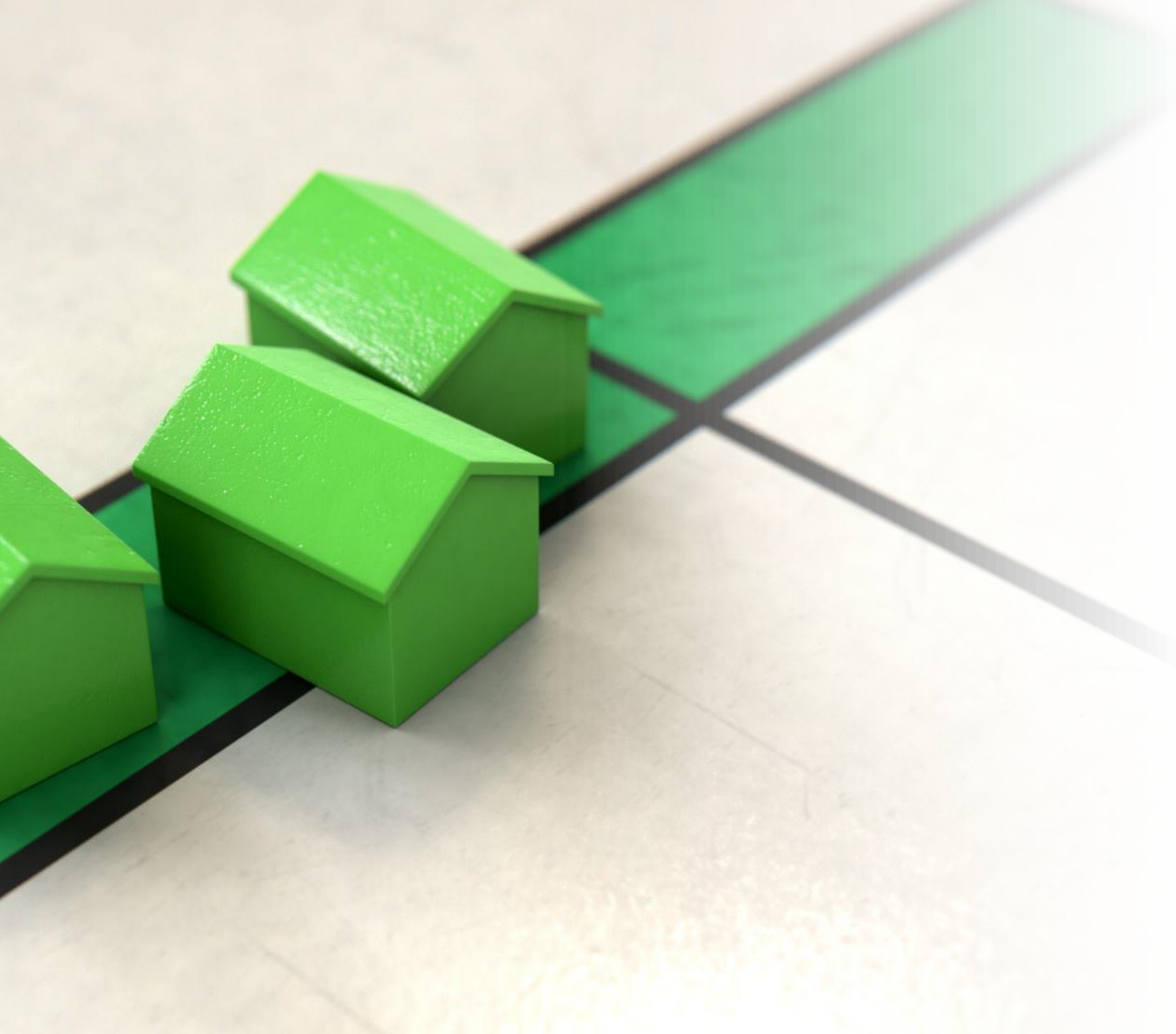
Controls Maximum Salience

	RuleTable BmiCalculation				
	CONDITION	ACTION	ACTION	DESCRIPTION	NAME
	$application: Application()				
	$param	System.out.println("Positive Risk Factor in application "+$application.getApplicationNumber() +". Premium adjustment: "+$param);	$application.addPremium($param);		
	Risk Factor	**Premium Adjustment**		**Comments**	**Rule Name**
	isOverweight()	10		10% increase for overweight BMI	Adjust premium for overweight client
	isObeseA()	20		20% increase for overweight BMI	Adjust premium for Obese Class A client
	isObeseB()	30		30% increase for overweight BMI	Adjust premium for Obese Class B client
	isObeseC()	40		40% increase for overweight BMI	Adjust premium for Obese Class C client
	isSmoker()	15		15% increase for overweight BMI	Adjust premium for smokers
	risk == "high"	30		30% increase for overweight BMI	Adjust premium for high-risk client
	risk == "medium"	15		15% increase for overweight BMI	Adjust premium for medium-risk client
	clientAge >= 70	20		20% increase for overweight BMI	Adjust premium for seniors
	PEC	20		20% increase for overweight BMI	Adjust premium for clients with PEC

	RuleTable IssueProposal				
	CONDITION	ACTION	ACTION	DESCRIPTION	NAME
	$application: Application()				
		System.out.println($1+" Final Premium Calculation: " + $2);	insert(new Proposal($1, $2));		
	Exists	**No Parameters Needed**		**Comments**	**Rule Name**
	TRUE	$application.getApplicationNumber(), $application.getPremium()		Issues proposal after adjustments	Issue insurance proposal

Module 07: Key Takeaways

1. Spreadsheet decision tables allow for the definition of business rules in a tabular format that is familiar and accessible to non-technical users.
2. Spreadsheet tables are .xls, .xlsx. or .csv tables that can be compiled into DRL rules
3. They must be properly formatted and follow the syntax requirements to be correctly compiled.



DROOLS 8

MODULE 08:
FINAL CAPSTONE PROJECT

BUILDING PRACTICAL SKILLS

Presented by John Paul Franke

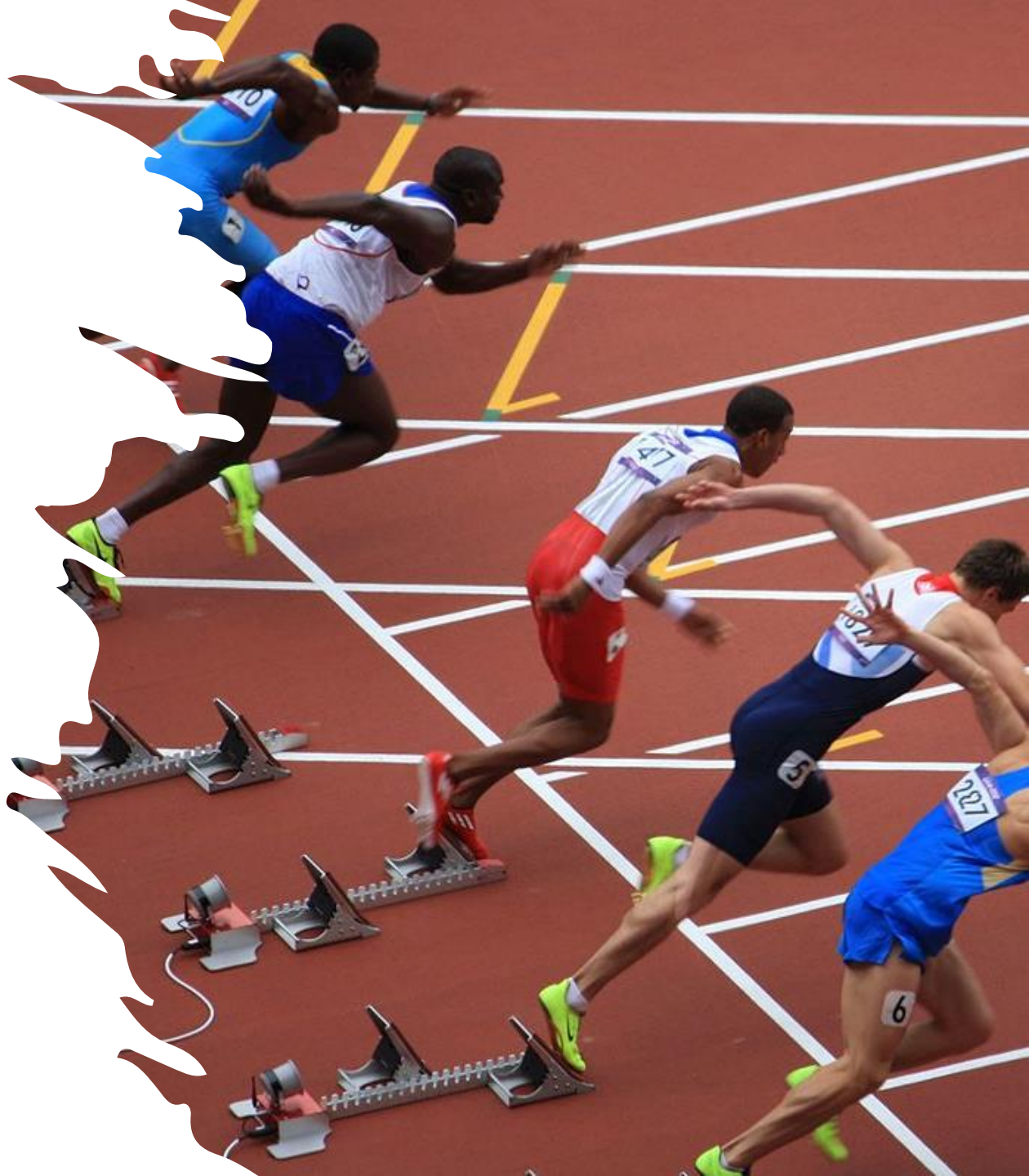
Capstone Project

Welcome to your final challenge of this course on Drools Rule Engine. It's time to put everything you've learned to practical use.

You will be designing and building a fully-functional drools maven project, cooperating with your fellow developers

Lab Exercise 03.02 - Final Capstone Project

Almost done! Open and Complete Lab Exercise 03.02 – Final Capstone Project



Thank You!