# Lab Exercise 02.02: Cross-Product Rule Challenge

**Objective:**

Create a Drools rule that invalidates any application where the `clientName` attribute matches the `clientName` of an already-existing policy. This exercise will help enhance understanding of how to write rules that cross-reference information between different objects. Additionally, students will modify the `ApplicationRepository` class to add a new `ApplicationBuilder` instance that intentionally triggers the newly created rule.

**Prerequisites:**

- Basic understanding of Drools rule Cross-Product syntax and structure.
- Access to the provided Java classes and repository setup.

## Instructions:

**Part 1: Writing the Rule**

1. **Identify Existing Policies**: Start by reviewing the `PolicyRepository` class to understand the structure of existing policies, paying close attention to the `clientName` attribute.
2. **Create the RuleFile**: In your Drools project, create a new rule file `InvalidApplicationRules.drl` within the `org.sw.lesson03.demo3` directory of your Maven project. (**Optional:** Comment out existing rules in other .drl files to reduce noise)
3. **Write the Rule**:
    - Define a package at the top of your `.drl` file.
    - Import dependency objects below package name.
    - Write a rule named `"Invalidate Applications with Existing Policy Names"`.
    - Use a `when` condition to search for an `Application` object that has a `clientName` matching any `clientName` from the existing policies in the `PolicyRepository`.
    - Use a `then` action to log a message indicating the application is invalid due to a name conflict with an existing policy and to change the application's status to `INVALID`.

***Part 2: Modifying ApplicationRepository***

1. Open the `ApplicationRepository.java` file within your project's `src/main/java` directory.
2. **Add a New ApplicationBuilder Instance**:
    - Create a new `ApplicationBuilder` instance with a `clientName` that matches one of the existing `clientNames` in the active policies.
    - Ensure the rest of the application's attributes are filled out to mimic a real application scenario.

**Example Addition**:

```
applications.add(Application.newBuilder()

    .withApplicationNumber("A-1010") // Ensure this number is uniq
ue

    .withClientName("John Doe") // Match this name with an existin
g policy

    .withClientAge(30)

    .withEmployed(true)

    .withPEC(false)

    .withRisk("Low")

    .withBMI(24)

    .withSmoker(false)

    .withClientIdNumber("ID-110")

    .build());
```

***Part 3: Testing the Rule***

1. **Verify Rule Execution**: Run Session `Lesson 3, demo 3` and verify that the application with a conflicting `clientName` is correctly identified and marked as invalid. Check the console output for the log message defined in the `then` action of your rule.
2. **Submission**: Once you have successfully tested your rule, submit your `.drl` file and the modified `ApplicationRepository.java` file as part of your lab exercise deliverables.

**Evaluation Criteria:**

- Correctness of the rule to identify applications with client names matching existing policies.
- Successful modification of the `ApplicationRepository` to add a conflicting application.
- Proper execution and validation of the rule within the Drools session.