

# TECHNOLOGY



**Caltech** | **Center for Technology & Management Education**

**Full Stack Java Developer**

# TECHNOLOGY



## Cucumber



## Scenarios



# Learning Objectives

By the end of this lesson, you will be able to:

- 👁 Learn Scenario in terms of Cucumber testing
- 👁 Define the Scenario outline with examples
- 👁 List the types of Scenarios
- 👁 Understand environment variable in Cucumber and its uses



# A Day in the Life of a Full Stack Developer

You are working in an organization and have been assigned a project.

You are working on a Java-based application and have been asked to perform testing on the application. You are supposed to run the same scenario repeatedly.

To do so, you need to explore the Scenario, Scenario outline, and the types of scenario in detail.

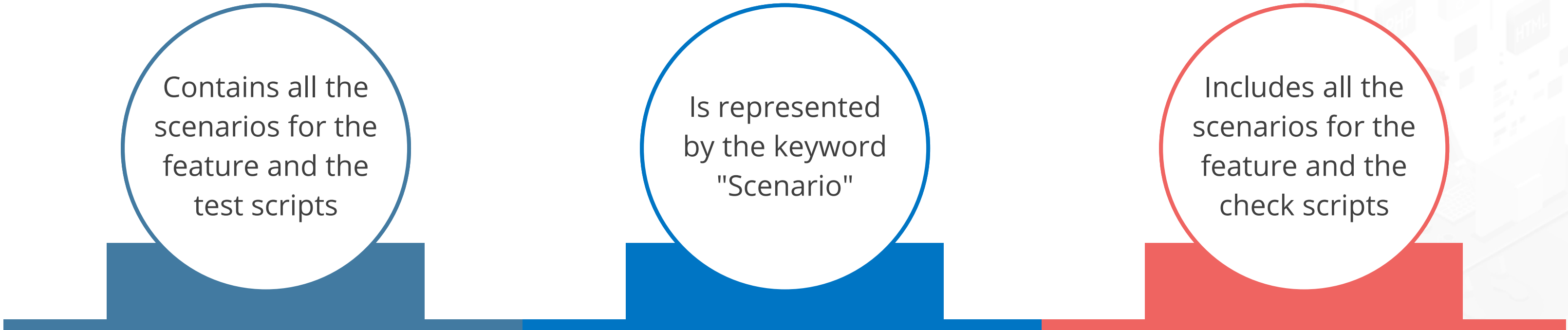




## Scenarios in Cucumber Testing

# Scenario in Cucumber Testing

The scenario is one of Gherkin's primary structures. It:



Contains all the scenarios for the feature and the test scripts

Is represented by the keyword "Scenario"

Includes all the scenarios for the feature and the check scripts

## Scenario: Example

Checking the feedback feature of a web application

**Scenario:** Validates the feedback functionality

**Given:** The user reaches xyz.com

**When:** The client taps on Feedback, at that time, the Feedback page opens.

**And:** The user submits a feedback message.

**Then:** The user's feedback should be displayed on the admin page

**And:** Administrator can answer the user

Every scenario uses the Given, When, And, and Then.



## Scenario Outline

# Scenario Outline

A scenario outline is used to repeatedly run the same scenario.



Scenario outlines are used when the identical test is performed numerous times with a different combination of values.

A scenario template is also referred to as a scenario outline.

## Scenario Outline: Example

**Case:** To test the login functionality for multiple customers

Use the scenario outline needed to execute the login capability several times



Use ">" for "Username" and "Password" in Gherkin sentences

# Scenario Outline: Example

**Case:** To test the login functionality for multiple customers

## Test script

**Scenario:** Successful login

**Given** user reaches the website xyz.com

**When** the user clicks on the login page

**And** user keys in the <username> in the **Login Window** as UserA

**And** <Password> as PassA

**Then** login is completed





## Scenario Outline: Example

**Case:** To test login functionality with various types of usernames and passwords

### Test script

**Feature:** Login Functionality

**Scenario Outline:** Login Functionality

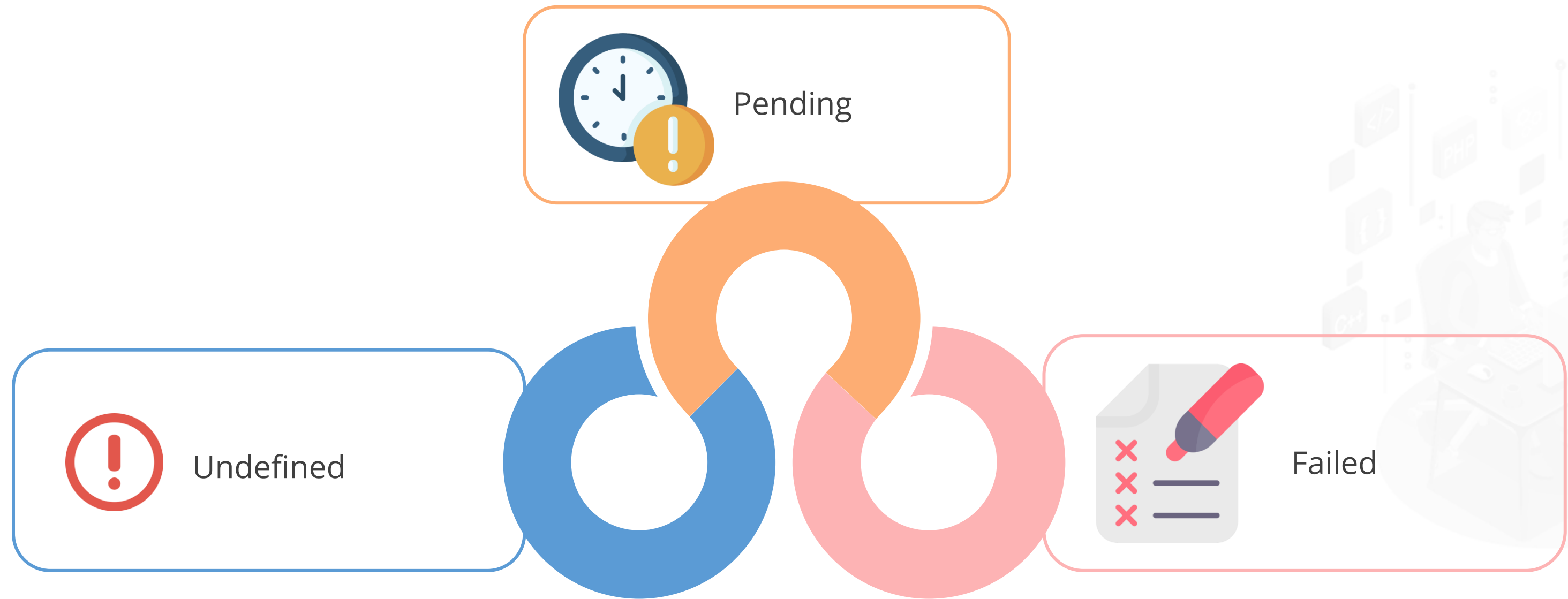
User		Pass
User1		PassA
User2		PassB
User3		PassC
User4		PassD

The cucumber will choose the variables with the given input value one-by-one.

Once the User has been executed with pass, the test will automatically pick the second iteration with another input value.

## Types of Scenarios

# Types of Scenarios



# Undefined Scenario

Undefined steps refer to the ones which are not discovered by Cucumber.

```
Running hellocucumber .RunCucumberTest
```

```
Feature: Is it Friday yet?
```

```
    Everyone wants to know when it's Friday
```

```
Scenario: Sunday isn't Friday
```

```
    #hellocucumber/is_it_Friday_yet.feature:4
```

```
    Given today is Sunday          #null
```

```
    When I ask whether it's Friday yet #null
```

```
    Then I should be told "Nope"      #null
```

```
Undefined scenario:
```

```
Hellocucumber/is_it_Friday_yet.feature:4  #Sunday isn't Friday
```

```
1 Scenario (1 undefined)
```

```
3 steps (3 undefined)
```

```
0m0.040s
```



# Undefined Scenario

Ensure that the path-to-door definitions (glue direction) are targeted correctly

Implement missing steps with the snippets below:

```
@Given("today is sunday")
Public void today_is_Sunday() {
    //Write code here that turns the phrase above into concrete
actions throw new io.cucumber.java.PendingException();
}

@When("I ask whether it's Friday yet")
Public void i_as_whether_it_s_Friday_yet() {
    //Write code here that turns the phrase above into concrete
actions throw new io.cucumber.java.PendingException();
}

@Then("I should be told {string}")
Public void i_should_be_told(String string) {
    //Write code here that turns the phrase above into concrete
actions throw new io.cucumber.java.PendingException();
}
```

# Undefined Scenario

Missing step mapping implies the scenario is undefined. Example:

When I add 4 and 5,

Then, at that point, the outcome is 9

And

the step "the outcome is 9" has a passing mapping

When

Cucumber executes the scenario

Then

the situation is indistinct

And

the step "the outcome is 9" is skipped



# Pending Scenario

The Scenario can be marked as pending to focus on other things.

To create a Pending step, add a step in one of the steps definition files.

```
Given /^PENDING/ do
  pending
end
```

Use it as:

Scenario: Login with valid credentials

Given PENDING, I have valid credentials.



# Failed Scenario

**Step 1:** In the Runner File inside the plugin, write "rerun: rerun/failed\_scenarios.txt"

```
import cucumber.api.CucumberOptions;
Import cucumber.api.junit.Cucumber;
Import org.junit.runner.RunWith;

@RunWith(Cucumber.class)
@CucumberOptions(
    feature = {"C:\\User\\ankur.jain\\eclipse-
workspace\\Cucumber_Test\\src\\main\\java\\com\\qa\\FeatureFile\\Login.feature"}
    //the path of the feature files
    //format={"html:target/site/cucumber.pretty;Jason:target/cucumber.Jason&quot"},

    Glue={"com.qa.stepDefinition"},
    //the path of the step definition files

    Plugin={"pretty","html:target/site/cucumber-
pretty","json:target/cucumber/cucumber.json","rerun:rerun/failed_scenario.txt"},
    //to generate different types of reporting

    Monochrome = true, //display the console output in a proper readable format
    Strict = true, //it will check if any step is not defined in step definition file
    dryRun = false //to check the mapping is proper between feature file and step def file //tags={"@all"}
)
Public class TestRunner
{}
```



# Failed Scenario

**Step 2:** Create another Runner File and name it as ReRunRunner.Java. Inside the feature, add the name of the failed\_scenario.txt file along with @

```
import cucumber.api.CucumberOptions;
import cucumber.api.junit.Cucumber;
import org.junit.runner.RunWith;

@RunWith(Cucumber.class)
@CucumberOptions(
    feature = {"@rerun/failed_scenario.txt"}
    //Cucumber execute failed scenario from this file
    //format={"html:target/site/cucumber-pretty;Jason:target/cucumber.Json""},

    Glue={"com.qa.stepDefinition"}, //the path of the step definition files

    Plugin={"pretty","html:target/site/cucumber-pretty","json:target/cucumber/cucumber.json","junit:target/cucumber.xml","com.cucumber.listener.ExtentCucumberFormatter:target/html/ExtentReport.html"}, //to generate different types of reporting

    Monochrome = true, //display the console output in a proper readable format
    Strict = true, //it will check if any step is not defined in step

    definition file
    dryRun = false //to check the mapping is proper between feature file and step def file //tags={"@all"}
)
Public class ReRunRunner
{}
```

## Failed Scenario

If a scenario fails, a failed scenarios.txt is created in the rerun folder.



Using the Java file, a failed scenario is rerun in the Cucumber-Junit.



## Environment Variables

# Environmental Variables

Cucumber uses an environmental variable to enable features.



## Note

Environment variables cannot be defined globally for security reasons.





# Environmental Variables

Users of Unix-based operating systems should not be placed in `/.bashrc`, `/.bash` profile, `/.zshrc`, `/etc.profile`.



## Note

Windows users should not be declared in the control panel or `setx.exe`-application.



## Refactoring in Cucumber

# Refactoring

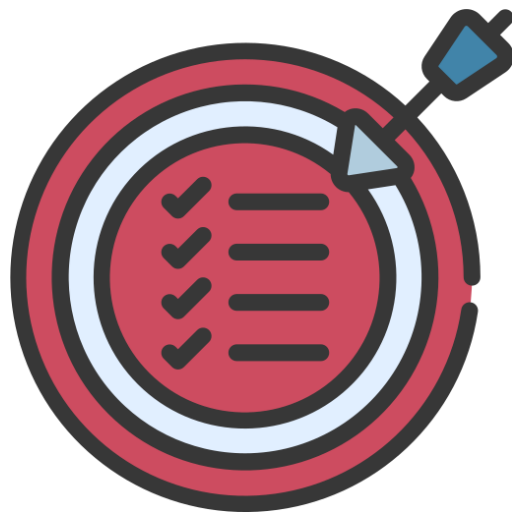
Refactoring is the process of organizing all the step definition files into groups.

Cucumber evaluates <any-extension> ending in \* inside the feature folder.



# Best Practices to Refactor

Group step definitions



Step definition files are recommended to be grouped based on each domain concept.

The thumb rule is to have one file for each domain object.





# Best Practices to Refactor

Write step definitions

Avoid including the steps that are absent in any of the scenarios in the step definition file.

Always implement step definitions that are required.

# Best Practices to Refactor

## Avoid Duplication



Always avoid writing similar step definitions, as they may create confusion.

It is advisable to execute helper methods for abstracting while documenting the steps.

# Avoid Duplication: Example

Using the helper method allows reducing them into a single step.

- Given I go to the Dashboard page
- Given I check the first page of the website
- Given I get the phone number and email to communicate

Helper method



Given I go to the {} page

The step definitions file:

```
@Given("I go to the {string} page")
public void navigateToWebPage(String page) {
    webpageFactory.openPage(page);
}
```

## Key Takeaways

- A Scenario outline is used to repeatedly run the same scenario.
- Undefined steps refer to the one which is not discovered by Cucumber.
- The scenario can be marked as pending to focus on other things.
- If a scenario fails, a failed scenarios.txt is created in the rerun folder.
- Refactoring is the process of organizing all the step definition files into groups.



# TECHNOLOGY

**Thank You**