Cucumber - Feature file

A **feature file** is a key part of **Behavior-Driven Development (BDD)**, especially when using tools like **Cucumber**, **SpecFlow**, or **Behave**.

Let's break it down from scratch:

🔰 What is a Feature File?

A **feature file** is a **plain text file** (usually with `.feature` extension) that contains a **human-readable description of software behavior**.

It's written using a language called **Gherkin**.

🔤 What is Gherkin?

Gherkin is a simple, structured language that uses **keywords like**:

- `Feature`
- `Scenario`
- `Given`
- `When`
- `Then`
- `And`, `But`

It allows **non-programmers** (like testers, product owners) to understand and contribute to test cases.

🧠 Purpose of a Feature File

- **Describe application features or functionalities** in simple language
- **Bridge the gap** between business people and developers/testers
- Help in **automating acceptance tests** using tools like Cucumber

---

🧾 Basic Structure of a Feature File

```gherkin
Feature: Login functionality

  Scenario: Successful login with valid credentials
    Given the user is on the login page
    When the user enters valid username and password
    And clicks on the login button
    Then the user should be redirected to the dashboard
```

Explanation:

- **Feature**: Describes the feature under test (e.g., Login functionality)
- **Scenario**: A concrete example/use case
- **Given**: Precondition
- **When**: Action/event
- **Then**: Expected outcome
- **And**: Additional steps

🔧 Tools That Use Feature Files

- **Cucumber** (Java, Ruby, etc.)
- **SpecFlow** (.NET)
- **Behave** (Python)
- **JBehave** (Java)

📌 Naming and Location

- File name: `login.feature`, `signup.feature`, etc.
- Typically located in a folder like `src/test/resources/features/` in Maven/Gradle projects

✅ Benefits

- Clear and easy communication
- Reduces ambiguity
- Helps build automated tests
- Encourages collaboration between developers, testers, and business analysts

🧪 Example with Automation (Cucumber + Java)

Step definition (Java code):

```java
@Given("the user is on the login page")
public void userOnLoginPage() {
    // code to open login page
}

@When("the user enters valid username and password")
public void enterCredentials() {
    // code to enter username and password
}
```

```java
@Then("the user should be redirected to the dashboard")
public void verifyDashboard() {
    // code to verify dashboard
}
```

The Gherkin steps in the `.feature` file are **mapped to Java methods** like these using annotations (`@Given`, `@When`, `@Then`).

Would you like a **mini-project using feature files in Cucumber with Java**?

Structuring a **feature file** properly is essential for **readability**, **maintainability**, and **effective test automation**. Here's a **step-by-step guide** to structure a feature file for beginners:

✅ 1. Start with the `.feature` Extension

The file should have a `.feature` extension, for example:

```pgsql
login.feature
searchProduct.feature
addToCart.feature
```

✅ 2. Begin with the `Feature:` Keyword

This describes the **high-level functionality** being tested.

```gherkin
Feature: Login functionality
  As a registered user
  I want to log into the system
  So that I can access my dashboard
```

> Optional lines after `Feature:` can describe the business value in simple terms (like a user story).

✅ 3. Add One or More `Scenario:` Blocks

Each `Scenario:` defines a **test case** (a specific example or use case).

```gherkin
Scenario: Successful login with valid credentials
```

Keep each scenario focused and small.

✅ 4. Write Steps Using Gherkin Syntax

Steps describe the behavior using these keywords:

- `Given`: Precondition
- `When`: Action
- `Then`: Outcome
- `And`, `But`: For additional steps

```
gherkin

    Given the user is on the login page
    When the user enters valid username and password
    And clicks on the login button
    Then the user should be redirected to the dashboard
```

✅ 5. Use `Scenario Outline` for Data-Driven Tests

If you want to run the same scenario with different sets of data:

```
gherkin

Scenario Outline: Login with multiple users
  Given the user is on the login page
  When the user enters "<username>" and "<password>"
  Then the user should see the "<status>" message

  Examples:
    | username | password | status        |
    | user1    | pass123  | success       |
    | user2    | wrongpwd | invalid login |
```

✅ 6. Optional: Use Tags for Filtering

Tags help group and run specific scenarios:

```
gherkin

@smoke
Feature: Login functionality

  @positive
  Scenario: Valid login
    ...

  @negative
  Scenario: Invalid login
    ...
```

🔖 Sample Full Feature File

```
gherkin

@smoke @login
Feature: Login to the application
  As a user of the application
  I want to be able to login
  So that I can access my dashboard

  @positive
  Scenario: Successful login
    Given the user is on the login page
    When the user enters valid username and password
    And clicks on the login button
    Then the user should be redirected to the dashboard
  @negative
  Scenario: Login with invalid password
    Given the user is on the login page
    When the user enters a valid username and wrong password
    Then an error message should be displayed
```

📁 Where to Place Feature Files in Your Project

In a Maven/Gradle project using Cucumber + Java:

```pgsql
project-root/
└── src/
    └── test/
        └── resources/
            └── features/
                ├── login.feature
                ├── registration.feature
                └── search.feature
```

Would you like me to show how this connects with step definition code in Java or another language?

Here are the **naming conventions** for feature files, scenarios, and steps in **BDD** using **Cucumber and Gherkin**, especially for beginners:

📁 1. Feature File Naming (File Name)

- **Use lowercase letters**
- **Separate words with underscores** or hyphens (_ or `-`)
- **Use `.feature` extension**
- Keep names **descriptive but short**

✅ Examples:

```pgsql
login.feature
user_registration.feature
add_to_cart.feature
checkout_payment.feature
```

❌ Avoid:

```
LoginTest.feature
Feature1.feature
testLogin.feature
```

🔸 Stick to functional descriptions, not test-centric terms like "Test" or "Case".

🧩 2. Feature Block Naming

Use the **`Feature:`** keyword followed by a **clear and concise title**.

✅ Good:

```gherkin
Feature: User login functionality
Feature: Product search by keyword
```

❌ Bad:

```gherkin
Feature: Test login page
Feature: Checking buttons
```

📝 Optionally include a **business goal** or **user story**:

```gherkin
Feature: User login functionality
  As a registered user
  I want to log in
  So that I can access my profile
```

📘 3. Scenario Naming

Use the `Scenario:` keyword followed by a **descriptive, plain-language title**:

✅ Good:

```gherkin
Scenario: Successful login with valid credentials
Scenario: Show error for wrong password
Scenario: Search returns results for valid keyword
```

❌ Bad:

```gherkin
Scenario: TC01
Scenario: test login
Scenario: Scenario1
```

- 🔸 Make it **readable and meaningful** to non-technical stakeholders too.

🔤 4. Step Naming (`Given`, `When`, `Then`, `And`)

- Write in **present tense**.
- Use **consistent vocabulary**.
- Avoid implementation details (like `click("#btn-login")`).

✅ Good:

```gherkin
Given the user is on the login page
When the user enters valid credentials
Then the user should be redirected to the dashboard
```

❌ Bad:

```gherkin
Given I launch the browser and go to the login page
When I input username and password in fields
Then I verify dashboard loaded
```

- 🔸 Steps should describe behavior, **not how it's done technically**.

🏷️ 5. Tag Naming

- Tags start with `@`
- Use lowercase
- Describe purpose, like `@smoke`, `@regression`, `@login`

✅ Examples:

```gherkin
gherkin

@smoke @login
Feature: Login feature
```

🔁 Summary

Element	Convention	Example
Feature File Name	lowercase, underscores, `.feature` extension	`user_login.feature`
Feature Name	Functional title, optional user story	`Feature: Search products`
Scenario Name	Describe expected behavior in plain English	`Scenario: Add item to cart`
Steps	Use Gherkin language in business terms	`Given the user is on homepage`
Tags	lowercase, functional, grouped	`@regression @checkout`

Would you like a **template with a properly named feature file + matching Java step definitions** next?

Great—writing multiple scenarios in one feature file is normal and encouraged, as long as they're **related to the same feature**. Here's how to do it cleanly, with best practices and examples.

---

1. When to put multiple scenarios together

You group **scenarios** in one **`.feature`** file when they all exercise different behaviors/examples of the **same high-level feature**.
Example feature: **`Login functionality`** can have:

- Successful login
- Login with wrong password
- Locked-out user
- Password reset flow (if closely tied)

---

2. Basic structure with multiple scenarios

```
gherkin

Feature: Login to the application
  As a registered user
  I want to log in
  So that I can access my dashboard

  @positive
  Scenario: Successful login with valid credentials
    Given the user is on the login page
    When the user enters valid username and password
    And clicks on the login button
    Then the user should be redirected to the dashboard

  @negative
  Scenario: Login fails with invalid password
    Given the user is on the login page
    When the user enters a valid username and wrong password
    Then an error message "Invalid credentials" should be displayed

  @edgecase
  Scenario: Login attempt with locked-out user
    Given the user account is locked
    And the user is on the login page
    When the user enters valid credentials
    Then a message "Account locked" should be shown
```

---

3. Using **`Background`** for shared preconditions

If multiple scenarios share the same setup, extract it into a **`Background`** block so you don't repeat it:

```
gherkin

Feature: Login to the application

  Background:
    Given the user is on the login page

  Scenario: Successful login
```

```
    When the user enters valid username and password
    And clicks on the login button
    Then the user should be redirected to the dashboard

  Scenario: Login fails with invalid password
    When the user enters a valid username and wrong password
    Then an error message should be displayed
```

***`Background`** runs before each scenario in the file.*

---

4. Data-driven variations with `Scenario Outline`

If you have the same scenario logic but different inputs/outcomes, use a **`Scenario Outline`** with **`Examples`**:

```gherkin
Scenario Outline: Login attempts with various credentials
  Given the user is on the login page
  When the user enters "<username>" and "<password>"
  Then the login result should be "<result>"

  Examples:
    | username | password  | result           |
    | user1    | correct1  | success          |
    | user2    | wrong     | invalid password |
    | user3    | locked    | account locked   |
```

---

5. Tagging for selective execution

You can tag individual scenarios or the entire feature, e.g.:

```gherkin
@smoke
Feature: Login to the application

  @critical
  Scenario: Successful login
    ...

  @regression
  Scenario: Login with wrong password
    ...
```

You can then run only **`@smoke`** or exclude **`@regression`** when executing tests.

---

6. Best practices

- **Keep scenarios independent**: One scenario's state shouldn't rely on another's execution order.
- **Avoid duplication**: Use **`Background`** for common setup; use step reuse.
- **Readable titles**: Scenario titles should describe behavior/outcome.
- **Limit scenario size**: One logical behavior per scenario.
- **Use meaningful tags**: For grouping like **`@smoke`**, **`@login`**, **`@negative`**, etc.

7. Example full feature with mixed scenarios

```
gherkin

@auth @login
Feature: User authentication

  Background:
    Given the authentication service is up
    And the user is on the login page

  @positive
  Scenario: Login with valid credentials
    When the user enters valid username and password
    Then the user should see the dashboard

  @negative
  Scenario: Login with incorrect password
    When the user enters valid username and wrong password
    Then an error message "Invalid credentials" is displayed

  @edgecase
  Scenario Outline: Multiple failed login attempts lock account
    When the user attempts login with "<username>" and wrong password "<password>"
    Then the account status should be "<status>"

    Examples:
      | username | password | status        |
      | user1    | badpass  | locked out    |
      | user1    | badpass  | locked out    |

  @forgot_password
  Scenario: User requests password reset
    When the user clicks "Forgot password"
    And enters their registered email
    Then a password reset link should be sent
```

8. Step definition mapping (example in Java/Cucumber)

```
java

@Given("the user is on the login page")
public void onLoginPage() { /* open login URL */ }

@When("the user enters valid username and password")
public void enterValidCredentials() { /* fill and submit */ }

@Then("the user should see the dashboard")
public void verifyDashboard() { /* assert dashboard visible */ }
```

Use parameterized steps for outlines:

```
java

@When("the user attempts login with {string} and wrong password {string}")
public void failedLogin(String username, String password) { /* attempt login */ }

@Then("the account status should be {string}")
public void checkStatus(String status) { /* verify locked/outcome */ }
```

Would you like a **ready-made `.feature` file** plus corresponding **step definition stubs in Java** to copy into your project?