Cucumber - Step definition

In **Cucumber**, a **Step Definition file** is where you write the actual **code implementation** for the steps written in your **feature files**.

🧩 Understanding the Basics

- A **Feature File** in Cucumber contains **Gherkin** syntax:

```pgsql
Feature: Login functionality
  Scenario: Successful login with valid credentials
    Given User is on login page
    When User enters valid username and password
    Then User should be redirected to the homepage
```

- But Gherkin steps like **`Given User is on login page`** don't do anything by themselves.
- The **Step Definition File** provides the Java (or other language) code that gets executed when Cucumber sees that step.

📄 Example of Step Definitions in Java

```java
import io.cucumber.java.en.Given;
import io.cucumber.java.en.When;
import io.cucumber.java.en.Then;

public class LoginSteps {

    @Given("User is on login page")
    public void user_is_on_login_page() {
        System.out.println("Navigated to login page");
    }

    @When("User enters valid username and password")
    public void user_enters_valid_credentials() {
        System.out.println("Entered username and password");
    }

    @Then("User should be redirected to the homepage")
    public void user_should_be_redirected_to_homepage() {
        System.out.println("Redirected to homepage");
    }
}
```

🔗 How Step Definitions Work

- **Cucumber matches** the step in the feature file to a Java method based on the string inside the annotation:
    - **`@Given("User is on login page")`** matches **`Given User is on login page`**

- When that step runs, the method associated with it is executed.

📌 Key Points

Concept	Details
Purpose	Bridge between feature file steps and actual test code
Annotations	**`@Given`**, **`@When`**, **`@Then`**, **`@And`**, **`@But`**
Location	Usually inside **`src/test/java/stepdefinitions`** folder
Language	Can be Java, Python, Ruby, etc., depending on setup

✅ Best Practices

- Keep each step concise and reusable.
- Avoid duplicating step definitions.
- Organize related steps into separate files (e.g., **`LoginSteps.java`**, **`RegisterSteps.java`**).

Would you like a complete runnable example of feature file + step definition using Selenium?

✅ Mapping Gherkin Steps to Java Methods in Cucumber

Mapping **Gherkin steps** to **Java methods** means connecting each line in the `.feature` file (written in natural language using Gherkin syntax) to a Java method that **executes the logic** behind that step.

🔹 1. Gherkin Step Example (Feature File)

```
gherkin

Feature: Login functionality

  Scenario: Successful login with valid credentials
    Given User is on the login page
    When User enters username "admin" and password "admin123"
    Then User should see the dashboard
```

🔹 2. Step Definition in Java (Mapping)

To map the above steps, you create Java methods using **annotations**:

```
java

import io.cucumber.java.en.*;

public class LoginStepDefinitions {

    @Given("User is on the login page")
    public void user_is_on_login_page() {
        // Code to navigate to login page
        System.out.println("Navigated to login page");
    }

    @When("User enters username {string} and password {string}")
    public void user_enters_credentials(String username, String password) {
        // Code to enter credentials
        System.out.println("Entered Username: " + username);
        System.out.println("Entered Password: " + password);
    }

    @Then("User should see the dashboard")
    public void user_should_see_dashboard() {
        // Code to verify dashboard visibility
        System.out.println("Dashboard is visible");
    }
}
```

🔸 Explanation of the Mapping

Gherkin Step	Java Method Annotation
`Given User is on the login page`	`@Given("User is on the login page")`
`When User enters username "admin"...`	`@When("User enters username {string} and password {string}")`
`Then User should see the dashboard`	`@Then("User should see the dashboard")`

- `{string}` is a **Cucumber expression** to match string values from Gherkin.

- These values are passed as parameters to the Java method.

🛠️ Supported Placeholders in Cucumber Expressions

Placeholder	Description	Example in Gherkin
`{string}`	Matches text in double quotes	`When user enters "admin"`
`{int}`	Matches an integer	`Then balance should be 100`
`{float}`	Matches a float	`Then tax should be 12.5`
`{word}`	Matches a single word	`Given user is {word}`

✅ Tips

- Annotations can use **regular expressions** or **Cucumber Expressions**.
- Always use **exact matching** strings unless you use parameters like `{string}`, `{int}`.
- Reuse steps across scenarios by using **parameterized step definitions**.

Would you like me to generate a complete example using Selenium with this mapping?

🧠 Using Regular Expressions in Step Definitions (Cucumber + Java)

In Cucumber, you can use **regular expressions (regex)** to create **flexible, dynamic step definitions** that can handle a variety of input values.

🔸 Gherkin Step with Parameters (Feature File)

```gherkin
gherkin

Scenario: Login with dynamic credentials
  Given User logs in with username "admin123" and password "pass@123"
```

🔹 Step Definition Using Regular Expressions

```java
java

import io.cucumber.java.en.Given;

public class LoginSteps {

    @Given("^User logs in with username \"([^\"]*)\" and password \"([^\"]*)\"$")
    public void user_logs_in_with_credentials(String username, String password) {
        System.out.println("Username: " + username);
        System.out.println("Password: " + password);
        // Add Selenium code to input and submit login form
    }
}
```

🔍 Explanation of Regex Pattern

Part	Meaning
`^`	Start of the line
`User logs in with username`	Matches static text
`"([^\"]*)"`	Captures everything inside quotes (non-greedy)
`and password`	More static text
`"([^\"]*)"`	Another capture group
`$`	End of the line

This pattern:

```regex
regex

^User logs in with username "([^\"]*)" and password "([^\"]*)"
```

matches:

```pgsql
pgsql

User logs in with username "admin123" and password "pass@123"
```

and captures:

- `admin123` into `username`
- `pass@123` into `password`

✅ Pros of Using Regex

- Handles **dynamic data**
- Supports **complex patterns** (e.g., email, number formats)
- Gives you **full control** over matching logic

⚠️ When to Use Regex vs Cucumber Expressions

Feature	Regular Expressions	Cucumber Expressions
Syntax	More powerful, but harder to read	Cleaner and easier for most cases
Example	`"([^\"]*)"`	`{string}`
Use case	Complex parsing, custom formats	Most standard scenarios
Readability	❌ Harder	✅ Easier

🔁 Same Step with Cucumber Expression (Simpler Alternative)

```
java

@Given("User logs in with username {string} and password {string}")
public void user_logs_in_with_credentials(String username, String password) {
    System.out.println("Username: " + username);
    System.out.println("Password: " + password);
}
```

🔪 Summary

- Use **regular expressions** when:
  - You need complex matching
  - You want fine-grained control
- Prefer **Cucumber Expressions** (`{string}`, `{int}`) for:
  - Simplicity
  - Better readability

Would you like an example using **regex with numbers or emails** in a Cucumber scenario?

✅ `@Given`, `@When`, `@Then` Annotations in Cucumber (Java)

In Cucumber, **`@Given`, `@When`, `@Then`** annotations are used to **map Gherkin steps** from your **`.feature`** file to **Java methods** in your step definition files.

🔹 1. `@Given`: Setup/Pre-condition

- Describes the **initial context or setup** for the scenario.
- Example: Opening a browser, navigating to a login page.

```
gherkin

Given User is on the login page
```

```
java

@Given("User is on the login page")
public void user_is_on_login_page() {
    System.out.println("User navigates to login page");
    // e.g., driver.get("https://example.com/login");
}
```

🔹 2. `@When`: Action/Trigger

- Describes the **event or action** performed by the user.
- Usually includes form submissions, clicking buttons, etc.

```
gherkin

When User enters valid username and password
```

```
java

@When("User enters valid username and password")
public void user_enters_credentials() {
    System.out.println("Entering credentials...");
    // e.g., driver.findElement(...).sendKeys("admin");
}
```

🔹 3. `@Then`: Outcome/Result

- Describes the **expected result or output**.
- Used for assertions and verifying UI elements, messages, redirects, etc.

```
gherkin

Then User should be redirected to the dashboard
```

```java
@Then("User should be redirected to the dashboard")
public void user_should_see_dashboard() {
    System.out.println("Verifying dashboard...");
    // e.g., Assert.assertTrue(driver.getTitle().contains("Dashboard"));
}
```

🔸 Bonus: `@And`, `@But`

- These are also supported and can be mapped to **`@Given`**, **`@When`**, or **`@Then`**:

  ```gherkin
  Given User is on login page
  And User has a valid account
  ```

  Both **`And`** and **`But`** can be annotated in Java using any of the main three (**`@Given`**, **`@When`**, **`@Then`**) depending on context.

✅ Summary Table

Gherkin Keyword	Java Annotation	Purpose
`Given`	**`@Given`**	Precondition or setup
`When`	**`@When`**	Action or user interaction
`Then`	**`@Then`**	Expected result or assertion
`And`/`But`	Depends on context	Additional step (same type)

Would you like a real-world Selenium + Cucumber example using all three annotations?

🔁 Reusing Step Definitions Across Scenarios in Cucumber (Java)

In Cucumber, **step definitions are designed to be reusable**—you write the implementation for a step **once**, and it can be **used in multiple scenarios or even multiple feature files**.

✅ Why Reuse Step Definitions?

- **DRY Principle**: Don't Repeat Yourself.
- Promotes **cleaner**, more **maintainable**, and **modular** test code.
- Reduces redundancy and effort.

🔹 Example: One Step Used in Multiple Scenarios

🗂️ `login.feature`

```
gherkin

Feature: Login functionality

  Scenario: Successful login
    Given User is on the login page
    When User logs in with username "admin" and password "admin123"
    Then User should see the dashboard

  Scenario: Login with wrong password
    Given User is on the login page
    When User logs in with username "admin" and password "wrongpass"
    Then User should see an error message
```

🔹 Step Definition File (Java)

```
java

import io.cucumber.java.en.*;

public class LoginSteps {

    @Given("User is on the login page")
    public void user_is_on_login_page() {
        System.out.println("Navigated to login page");
    }

    @When("User logs in with username {string} and password {string}")
    public void user_logs_in(String username, String password) {
        System.out.println("Username: " + username);
        System.out.println("Password: " + password);
        // Selenium login code here
    }

    @Then("User should see the dashboard")
    public void user_should_see_dashboard() {
        System.out.println("Dashboard is visible");
        // Assert dashboard presence
    }

    @Then("User should see an error message")
    public void user_should_see_error_message() {
        System.out.println("Error message displayed");
        // Assert error message
```

```
    }
}
```

💡 How Reuse Works

Cucumber matches Gherkin steps **by the text string in the annotations**, **not by scenario**.

As long as the Gherkin step:

```
gherkin

Given User is on the login page
```

appears **exactly the same** in multiple scenarios, the same method will be used.

🛠️ Tips for Maximum Reuse

Tip	Example
Use **parameterization** (**`{string}`**, **`{int}`**)	Avoid writing multiple steps like **`User enters admin`**, **`User enters user`**
Keep step texts **generic**	**`User logs in with username {string}`** instead of **`Admin logs in`**
Organize steps into **logical files**	E.g., **`LoginSteps.java`**, **`SearchSteps.java`**
Avoid hard-coding	Use parameters or config for values

❌ What Not to Do

```
gherkin

Given Admin is on login page  # Bad: specific to Admin
Given User is on login page   # Good: reusable
```

✅ Summary

Reusability Principle	Description
Match by annotation string	Cucumber binds based on the string, not scenario
Use parameters	Allows flexible and dynamic step reuse
Organize by concern	Keep step files organized for maintainability

Would you like a reusable **Selenium login step** example with page object model (POM)?