Cucumber JVM explanation

give me the deep and detailed explaination of Cucumber JVM in java so that i can able to understand it well and do not have any type of doubt in this.

so give me the deep and detailed explaination

Alright Piyush 👍
I'll explain **Cucumber JVM** from **absolute basics** → **internal working** → **real project structure** → **execution flow** → **best practices**, in **simple language**, step by step, so that **no confusion remains**.

🌱 What is Cucumber JVM (in Java)?

Cucumber JVM is a **testing tool** that allows you to write **automated tests in plain English**, and then **connect those English steps to Java code**.

It follows **BDD (Behavior Driven Development)**.

👉 Instead of writing only Java test code, you write:

- **Human-readable test scenarios**
- Then **Java code executes those steps**

🔹 Why the name "Cucumber JVM"?

- **Cucumber** → the BDD testing framework
- **JVM** → it runs on the **Java Virtual Machine**

So:

> **Cucumber JVM = Cucumber framework + Java ecosystem**

It integrates smoothly with:

- JUnit
- TestNG
- Selenium, REST Assured, Spring Boot, etc.

🧠 Problem Cucumber Solves

❌ Traditional Testing Problem

```java
@Test
public void testLogin() {
    // lots of code
}
```

❌ Only developers understand
❌ Business people / QA / Product owners cannot read it

✅ Cucumber Solution

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

✔️ Readable by **non-technical people**
✔️ Executable by **Java code**

🧩 Core Concepts of Cucumber JVM

Let's break everything down.

1️⃣ BDD (Behavior Driven Development)

BDD focuses on:

- **Behavior of the application**
- **From user's perspective**
- Written in **simple language**

BDD answers:

- What should the system do?
- Under what conditions?
- What should be the result?

2️⃣ Gherkin Language (VERY IMPORTANT)

Cucumber scenarios are written using **Gherkin**.

👉 Gherkin is **not Java**, it's a **plain-text language**.

Keywords:

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

Example Feature File

```
gherkin

Feature: Login functionality

  Scenario: Successful login
    Given user is on login page
    When user enters valid username and password
    Then user should see dashboard
```

📌 This file is usually saved as:

```
pgsql

login.feature
```

Meaning of Keywords (Important!)

Keyword	Meaning
Feature	High-level functionality
Scenario	A test case
Given	Precondition
When	Action
Then	Expected outcome
And/But	Additional steps

3️⃣ Feature File (WHAT to test)

- Written in **.feature** files
- Contains **business-readable tests**
- No Java code here

📂 Location:

```swift
src/test/resources/features/
```

4️⃣ Step Definitions (HOW to test)

This is where **Java code lives**.

Each Gherkin step is **mapped to a Java method**.

Example Step Definition

```java
@Given("user is on login page")
public void user_is_on_login_page() {
    System.out.println("Opening login page");
}
```

Annotation Mapping:

Gherkin Step	Java Annotation
Given	`@Given`
When	`@When`
Then	`@Then`

🔥 IMPORTANT RULE

> **Each Gherkin step must match exactly ONE Java method**

If not:

- ❌ Undefined step error
- ❌ Ambiguous step error

5️⃣ How Cucumber Executes Tests (INTERNAL FLOW)

This is **very important for clarity**.

Execution Flow:

```mathematica
Feature File
    ↓
Scenario
```

```
    ↓
Steps (Given/When/Then)
    ↓
Matching Java Step Definition
    ↓
Java Code Executes
    ↓
Pass / Fail
```

Step Matching Logic

```gherkin
Given user is on login page
```

Cucumber looks for:

```java
@Given("user is on login page")
```

✔️ Exact match → executed
❌ No match → test fails

6️⃣ Regular Expressions & Parameters

Steps can take **dynamic values**.

Feature File:

```gherkin
When user enters username "admin" and password "1234"
```

Step Definition:

```java
@When("user enters username {string} and password {string}")
public void enterCredentials(String username, String password) {
    System.out.println(username + " " + password);
}
```

📌 `{string}`, `{int}`, `{double}` → placeholders

7️⃣ Cucumber Test Runner (HOW tests start)

Cucumber needs a **runner class**.

This class:

- Tells where feature files are
- Tells where step definitions are
- Configures reports

Example Runner Class (JUnit)

```java
@RunWith(Cucumber.class)
@CucumberOptions(
    features = "src/test/resources/features",
    glue = "stepdefinitions",
    plugin = {"pretty", "html:target/report.html"},
    monochrome = true
)
public class TestRunner {
}
```

📌 Uses:

- JUnit
- `@RunWith(Cucumber.class)`

8️⃣ Glue Code (VERY IMPORTANT)

```java
glue = "stepdefinitions"
```

Means:

> “Cucumber, search for step definition classes in this package.”

❌ Wrong glue → steps not found
✔️ Correct glue → steps executed

9️⃣ Hooks (Before & After)

Hooks are used to:

- Setup before scenario
- Cleanup after scenario

Example Hooks

```java
@Before
public void setup() {
    System.out.println("Before scenario");
}

@After
public void teardown() {
    System.out.println("After scenario");
}
```

Use cases:

- Open browser
- Close browser
- DB setup
- Cleanup test data

🔟 Tags (Selective Execution)

You can **tag scenarios**.

```
gherkin

@smoke
Scenario: Successful login
```

Run only tagged tests:

```
java

@CucumberOptions(tags = "@smoke")
```

✔️ Smoke testing
✔️ Regression testing
✔️ Fast execution

1️⃣1️⃣ Reporting in Cucumber

Cucumber provides:

- HTML reports
- JSON reports
- Extent reports (with plugins)

```
java

plugin = {
    "pretty",
    "html:target/cucumber-report.html",
    "json:target/cucumber.json"
}
```

1️⃣2️⃣ Integration with Selenium (Real Use Case)

Cucumber itself:
❌ Does NOT automate browser

It **controls tools like Selenium**.

Flow:

```css
Gherkin → Step Definition → Selenium Code → Browser
```

Example

```java
@Given("user opens browser")
public void openBrowser() {
    WebDriver driver = new ChromeDriver();
    driver.get("https://example.com");
}
```

1️⃣3️⃣ Folder Structure (BEST PRACTICE)

```bash
src/test/java
 ├── runner
 │    └── TestRunner.java
 ├── stepdefinitions
 │    └── LoginSteps.java
 └── hooks
      └── Hooks.java

src/test/resources
 └── features
      └── login.feature
```

1️⃣4️⃣ Common Errors (IMPORTANT)

❌ Undefined Step

- Step exists in feature
- No Java mapping

❌ Ambiguous Step

- Same step mapped to **multiple methods**

❌ Glue Issue

- Wrong package path

1️⃣5️⃣ Why Cucumber is Widely Used

✔️ Business-readable tests
✔️ Collaboration between Dev + QA + Business
✔️ Reusable steps
✔️ Clean separation of WHAT vs HOW
✔️ Excellent for automation frameworks

🧠 Final Mental Model (Remember This)

```css
Feature File  →  WHAT to test
Step Defs    →  HOW to test (Java)
Runner       →  HOW to run
Hooks        →  Setup & Cleanup
Tags         →  Control execution
```

If you want, next I can:

- 🔹 Explain **Cucumber + Selenium framework design**
- 🔹 Explain **Cucumber with Spring Boot**
- 🔹 Explain **Real interview questions on Cucumber**
- 🔹 Build a **complete end-to-end project structure**

Just tell me 👍