Cucumber BDD – Complete Practical Notes (Java / Spring Boot)

---

1. Why Cucumber Exists (Context in Testing Pyramid)

Testing Pyramid

- **Unit Tests** – JUnit, Mockito
- **Integration Tests** – @SpringBootTest, Testcontainers
- **Acceptance / BDD Tests** – **Cucumber**

Cucumber validates **business behavior**, not implementation.

> Cucumber answers: *"Does the system behave as the business expects?"*

---

2. BDD Fundamentals

Gherkin Keywords

- **Feature** – Business capability
- **Scenario** – Concrete example
- **Given** – Preconditions
- **When** – Action
- **Then** – Expected result
- **And / But** – Readability

Maps cleanly to: - Arrange → Act → Assert

---

3. Basic Cucumber Components

3.1 Feature File

```
Feature: User Login

  Scenario: Valid user can login
    Given user is on login page
    When user enters valid username and password
    Then user should be redirected to dashboard
```

Location:

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

3.2 Step Definitions

```java
@Given("user is on login page")
public void user_on_login_page() {
}

@When("user enters valid username and password")
public void user_enters_credentials() {
}

@Then("user should be redirected to dashboard")
public void verify_dashboard() {
}
```

Rules: - One step = one method - Assertions belong in **Then**

3.3 JUnit Runner

```java
@Cucumber
public class CucumberTest {
}
```

---

4. Integrating Cucumber with Spring Boot

Spring Context Configuration

```java
@CucumberContextConfiguration
@SpringBootTest
public class CucumberSpringConfig {
}
```

Benefits: - Dependency Injection - Same DB, security, config as integration tests

---

5. Reusing Unit & Integration Test Knowledge

Cucumber **calls your real services**.

```
When user places an order
```

```
@When("user places an order")
public void place_order() {
    orderService.placeOrder(order);
}
```

> Business flow verified, logic already unit-tested.

---

6. Cucumber with REST APIs

Using MockMvc

```
@Autowired
MockMvc mockMvc;

@When("client creates a hotel")
public void create_hotel() throws Exception {
    mockMvc.perform(post("/hotels")
        .contentType(MediaType.APPLICATION_JSON)
        .content("{\"name\":\"Hilton\"}"))
        .andExpect(status().isCreated());
}
```

Using RestAssured

```
RestAssured.given()
  .contentType(ContentType.JSON)
  .body(payload)
  .post("/hotels")
  .then()
  .statusCode(201);
```

---

7. Database Validation in Cucumber

```
@Autowired
HotelRepository hotelRepository;

@Then("hotel should exist in database")
public void verify_db() {
    assertTrue(hotelRepository.findByName("Hilton").isPresent());
}
```

Use same DB strategy as integration tests: - H2 - Testcontainers

---

8. Scenario Outline (Parameterized Tests)

```
Scenario Outline: Login validation
  When user logs in with "<username>" and "<password>"
  Then login should be "<result>"

Examples:
  | username | password | result |
  | admin    | admin123 | success |
  | admin    | wrong    | failure |
```

---

9. Hooks – Setup and Cleanup

```
@Before
public void setup() {
    // reset DB, init data
}

@After
public void cleanup() {
    // cleanup
}
```

Use cases: - Reset DB state - Auth token setup - Test data preparation

---

10. Tagging Scenarios

```
@smoke
Scenario: Login success
```

Run selectively:

```
mvn test -Dcucumber.filter.tags="@smoke"
```

---

11. Parallel Execution

```
cucumber.execution.parallel.enabled=true
cucumber.execution.parallel.config.strategy=fixed
cucumber.execution.parallel.config.fixed.parallelism=4
```

Precautions: - Isolated DB state - No shared static data

---

12. Best Practices

DO

- Keep scenarios business-readable
- Reuse step definitions
- Validate behavior, not internals

DON'T

- Put logic in step definitions
- Mock everything
- Write too many scenarios

---

13. Anti-Patterns

❌ One scenario = many assertions ❌ UI-level steps for API testing ❌ Using Cucumber instead of unit tests

---

14. Typical Project Structure

```
src/test
 ├── java
 │   ├── steps
 │   ├── config
 │   └── CucumberTest.java
 └── resources
     └── features
```

---

15. Execution Order in CI

1. Unit Tests
2. Integration Tests
3. Cucumber Tests

---

16. Interview-Ready Summary

- Cucumber is for **acceptance testing**
- Uses **Gherkin** for collaboration
- Integrates with **JUnit + Spring Boot**
- Complements unit & integration tests

---

17. When to Use Cucumber

✔️ Business workflows ✔️ Multi-service flows ✔️ Security rules

✘ Algorithms ✘ Utility methods ✘ Edge-case validation

---

18. Where Gatling Fits (Performance Testing Context)

Testing Pyramid (Extended)

- Unit Tests → correctness of logic
- Integration Tests → component interaction
- **Cucumber (BDD)** → business behavior & acceptance
- **Gatling** → performance, load, and scalability

  Gatling answers: *"How does the system behave under load?"*

19. Cucumber vs Gatling (Clear Comparison)

Aspect	Cucumber	Gatling
Purpose	Business behavior validation	Performance & load testing
Focus	Correctness	Throughput, latency, stability
Language	Gherkin + Java	Scala DSL
Data Volume	Small, readable	Large, realistic
CI Usage	Functional gate	Performance gate

---

20. How Cucumber, Integration Tests, and Gatling Work Together

Typical Pipeline

1. **Unit Tests** – Fast feedback
2. **Integration Tests** – Wiring & data
3. **Cucumber Tests** – Business acceptance
4. **Gatling Tests** – Load & stress

Failures mean: - Unit failure → logic bug - Integration failure → config / dependency issue - Cucumber failure → business rule broken - Gatling failure → scalability / performance risk

---

21. Reusing Cucumber Scenarios for Gatling (Conceptual)

Cucumber and Gatling **should not share code**, but they **share intent**.

Example: - Cucumber scenario validates: - User can place order - Gatling simulation validates: - 1,000 users can place orders concurrently

---

22. Basic Gatling Example (Conceptual)

```
class OrderSimulation extends Simulation {
  val httpProtocol = http.baseUrl("http://localhost:8080")

  val scn = scenario("Order Load Test")
    .exec(http("Create Order")
      .post("/orders")
      .body(StringBody("{ \"item\": \"TV\" }")).asJson)
```

```
  setUp(
    scn.inject(atOnceUsers(100))
  ).protocols(httpProtocol)
}
```

---

23. Common Anti-Pattern (Very Important)

❌ Using Cucumber for load testing ❌ Using Gatling for business validation

Each tool has a **non-overlapping responsibility**.

---

24. Interview-Ready One-Liner

- **JUnit** → verifies code correctness
- **Integration Tests** → verifies system wiring
- **Cucumber** → verifies business behavior
- **Gatling** → verifies system performance under load

End of Notes