Mocking in Java Explained

give me the deep and detailed explaination of Mocking 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

Below is a **deep, end-to-end explanation of Mocking in Java**, written **slowly, conceptually, and practically**, so that **no confusion remains**. I'll build everything from **first principles**, then move step-by-step into **real Java + Mockito usage**, exactly suited for a **backend / Spring Boot developer like you**.

🔷 What is Mocking in Java? (From Absolute Zero)

First, understand the problem mocking solves

When you write **unit tests**, your goal is to test **only one class at a time**.

But in real applications:

```text
Controller → Service → Repository → Database
```

Each class **depends on other classes**.

❓ **Problem**
If you test **`Service`**, it automatically calls:

- Database
- Network
- External APIs
- File system

This causes:

- Slow tests ❌
- Flaky tests ❌
- Hard-to-control behavior ❌

👉 **Mocking exists to break this dependency chain.**

🔹 Definition (Very Important)

Mocking means creating a **fake object** that:

- Looks like a real object
- Behaves in a **controlled way**
- Does **NOT execute real logic**

🔷 Why Mocking is REQUIRED (Not Optional)

Without mocking:

```java
OrderService → PaymentGateway → Bank Server ❌
```

You cannot:

- Test offline
- Control failures
- Predict responses

With mocking:

```java
OrderService → MockPaymentGateway ✅
```

Now you can:

- Decide what response comes
- Simulate failures
- Test logic safely

🔷 Real-Life Analogy (Crystal Clear)

Imagine testing a Car

Component	Real Version	Mock Version
Engine	Actual engine	Fake engine
Fuel	Real petrol	Dummy petrol
Speed	Actual speed	Controlled speed

You **don't test the engine** when testing:

> “Does the speedometer show speed?”

Same idea in Java.

🔷 Types of Test Doubles (IMPORTANT THEORY)

Mocking is part of **Test Doubles**.

1️⃣ Dummy

- Passed but never used

```java
null, empty object
```

2️⃣ Stub

- Returns fixed values

```java
when(repo.find()).thenReturn(data);
```

3️⃣ Fake

- Working but simplified

```java
In-memory DB
```

4️⃣ Mock ⭐ (Most important)

- Verifies **interaction**

```java
verify(repo).save();
```

🔷 Popular Mocking Frameworks in Java

Most used:

- **Mockito** ⭐⭐⭐⭐⭐

Others:

- **EasyMock**
- **PowerMock**

👉 In real industry **Mockito + JUnit** is standard.

🔷 Mockito + JUnit Architecture

Testing stack:

- **JUnit** → Test runner
- **Mockito** → Fake dependencies

🔷 Core Mockito Annotations (VERY IMPORTANT)

Annotation	Purpose
`@Mock`	Create fake object
`@InjectMocks`	Inject mocks into class
`@BeforeEach`	Setup
`@ExtendWith(MockitoExtension.class)`	Enable Mockito

🔷 Simple Example WITHOUT Mocking (Problem)

```java
class UserService {
    private UserRepository repo;

    public UserService(UserRepository repo) {
        this.repo = repo;
    }

    public String getUserName(int id) {
        return repo.findById(id).getName();
    }
}
```

Test ❌ (Bad)

```java
repo.findById() → DB connection → FAIL
```

🔷 SAME Example WITH Mocking (Correct Way)

Step 1️⃣ Create Mock

```java
@Mock
UserRepository userRepository;
```

Step 2️⃣ Inject into Service

```java
@InjectMocks
UserService userService;
```

Step 3️⃣ Define Behavior (Stubbing)

```java
when(userRepository.findById(1))
    .thenReturn(new User(1, "Piyush"));
```

👉 You **decide the output**.

Step 4️⃣ Call Method

```java
String name = userService.getUserName(1);
```

Step 5️⃣ Assert Result

```java
assertEquals("Piyush", name);
```

🔷 Complete Working Test Class

```java
@ExtendWith(MockitoExtension.class)
class UserServiceTest {

    @Mock
    UserRepository userRepository;

    @InjectMocks
    UserService userService;

    @Test
    void testGetUserName() {
        when(userRepository.findById(1))
            .thenReturn(new User(1, "Piyush"));

        String result = userService.getUserName(1);

        assertEquals("Piyush", result);
    }
}
```

✔️ No DB
✔️ No Spring context
✔️ Fast
✔️ Reliable

🔷 Understanding `when().thenReturn()` (Internals)

```java
when(mock.method()).thenReturn(value);
```

Means:

> “When this method is called on the mock, **don’t execute real logic**, just return this value.”

🔷 Verifying Interactions (CORE MOCKING FEATURE)

Mocks don’t just return values — they **track calls**.

```java
verify(userRepository).findById(1);
```

This checks:

- Method called?
- How many times?
- With which arguments?

Verify Call Count

```java
verify(repo, times(1)).save(user);
verify(repo, never()).delete(any());
```

🔷 Mock vs Spy (VERY CONFUSING — Explained Simply)

Feature	Mock	Spy
Real object	❌	✅
Calls real methods	❌	✅
Can stub	✅	✅

Spy Example

```java
List<String> list = spy(new ArrayList<>());
list.add("Java");
```

👉 Use **Spy** only when partial real behavior is needed.

🔷 Mocking Exceptions

```java
when(repo.findById(1))
    .thenThrow(new RuntimeException("DB error"));
```

Used to test:

- Failure paths
- Retry logic
- Error handling

🔷 Mocking Void Methods

```java
doNothing().when(repo).save(user);
doThrow(new Exception()).when(repo).delete(id);
```

🔷 Argument Matchers (VERY IMPORTANT)

```java
any()
anyInt()
anyString()
eq(5)
```

Example:

```
java

when(repo.findById(anyInt()))
    .thenReturn(user);
```

🔷 Mocking in Spring Boot

Spring uses **Dependency Injection**, so mocking fits naturally.

With Spring Boot:

```
java

@MockBean
UserRepository repo;
```

👉 Replaces real bean inside Spring context.

🔷 Unit Test vs Integration Test

Feature	Unit Test	Integration Test
Uses mocks	✅	❌
DB involved	❌	✅
Speed	Fast	Slow
Scope	Single class	Multiple layers

🔷 Common Mistakes (VERY IMPORTANT)

❌ Mocking the class under test
❌ Too many mocks
❌ Testing implementation instead of behavior
❌ Mocking simple POJOs

🔷 When NOT to Mock

❌ DTOs
❌ Entity classes
❌ Utility classes (most times)

🔷 Mental Model (Remember This Forever)

> **Mocking is about controlling the environment, not faking logic.**

🔷 Interview One-Line Definition (Perfect)

> “Mocking in Java is a technique used in unit testing to isolate a class by replacing its dependencies with controlled fake objects, allowing deterministic and fast tests.”

If you want next:

- 🔹 **Mocking vs Stubbing deep comparison**
- 🔹 **Mockito internal working**
- 🔹 **Mocking REST APIs**
- 🔹 **Real Spring Boot project example**
- 🔹 **Mockito interview questions**

Just tell me 👍