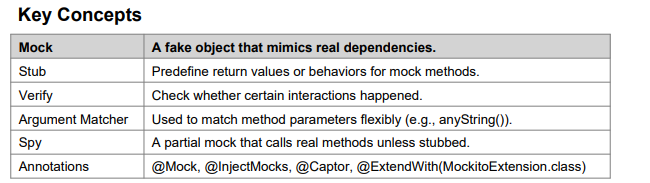
**Mockito :**

. Mockito is a popular Java mocking framework used to write unit tests by creating mock objects for dependencies. It allows developers to focus on business logic without involving real implementations such as databases, web services, or external APIs.



**Step 1 – Create a Maven Project in STS**

1. Open **STS** → File → New → Maven Project.
2. Check *Create a simple project (skip archetype selection)*.
3. Fill:
   * **Group Id:** com.example
   * **Artifact Id:** mockito-demo
   * Packaging: jar
4. Click Finish.

**Step 2 – Add Mockito & JUnit 5 dependencies in pom.xml**

<dependencies>

<!-- JUnit 5 -->

<dependency>

<groupId>org.junit.jupiter</groupId>

<artifactId>junit-jupiter</artifactId>

<version>5.10.2</version>

<scope>test</scope>

</dependency>

<!-- Mockito Core -->

<dependency>

<groupId>org.mockito</groupId>

<artifactId>mockito-core</artifactId>

<version>5.12.0</version>

<scope>test</scope>

</dependency>

<!-- Mockito with JUnit 5 Integration -->

<dependency>

<groupId>org.mockito</groupId>

<artifactId>mockito-junit-jupiter</artifactId>

<version>5.12.0</version>

<scope>test</scope>

</dependency>

<!-- Optional: For mocking final/static methods -->

<dependency>

<groupId>org.mockito</groupId>

<artifactId>mockito-inline</artifactId>

<version>5.2.0</version>

<scope>test</scope>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-surefire-plugin</artifactId>

<version>3.2.5</version>

</plugin>

</plugins>

</build>

**Tip:** After adding, **Right-click on project → Maven → Update Project** in STS.

**Step 3 – Enable JUnit 5 in STS**

* **Right-click → Properties → Java Build Path → Libraries**
* Ensure JUnit 5 is available.
* If you only see JUnit 4, add JUnit Jupiter from Maven.

**Step 4 – Ready to Write Mockito Tests**

You can now write tests in the src/test/java folder using @ExtendWith(MockitoExtension.class).

## 1) Create a test package

In **Package Explorer**, expand your project

Right-click src/test/java **→ New → Package`**.

Enter a package name, e.g., com.example.

Click **Finish**.

## 2) Create a JUnit 5 test class

Right-click src/test/java/com.example **→ New → Other…**.

Type **“junit”** in the filter → choose **JUnit Test Case** → **Next**.

**Name**: UserServiceTest (end with Test so Maven picks it up).

**Select JUnit 5** (very important).

Uncheck any setup templates if you don’t need them → **Finish**.

1. **Mock**
   * A fake object that imitates a real dependency.
   * Example: Mocking a database repository so you don’t hit the real DB.
2. **Stub**
   * Predefining what a mock should return when a method is called.
   * when(mock.method()).thenReturn(value);
3. **Verify**
   * Check if a certain method was called with expected arguments.
   * verify(mock).method(arg);
4. **Argument Matchers**
   * Flexible matching for parameters.
   * when(mock.method(anyString())).thenReturn("data");
5. **Spy**
   * A partial mock — wraps a real object but allows overriding methods.
6. **Annotations**
   * @Mock – Create a mock.
   * @InjectMocks – Inject mocks into the class under test.
   * @Captor – Capture arguments passed to methods.
   * @ExtendWith(MockitoExtension.class) – JUnit 5 integration.
7. **Why Use Mockito?**
   * Faster tests (no real DB, API calls).
   * Test in isolation.
   * Easier to simulate failures.

**Example (User Registration Service)**

**Business Scenario**

We have:

* UserRepository → stores users in DB.
* EmailService → sends a welcome email.
* UserService → business logic for registration.

**Classes**

1. User.java

package com.example;

public class User {

private Long id;

private String email;

private String name;

public User(String email, String name) {

this.email = email;

this.name = name;

}

// getters and setters

public Long getId() { return id; }

public void setId(Long id) { this.id = id; }

public String getEmail() { return email; }

public String getName() { return name; }

}

2. UserRepository.java

package com.example;

import java.util.Optional;

public interface UserRepository {

User save(User user);

Optional<User> findById(Long id);

}

3. EmailService.java

package com.example;

public class EmailService {

public void sendWelcomeEmail(User user) {

// Imagine sending email

}

}

4. UserService.java

package com.example;

public class UserService {

private final UserRepository repo;

private final EmailService emailService;

public UserService(UserRepository repo, EmailService emailService) {

this.repo = repo;

this.emailService = emailService;

}

public User register(String email, String name) {

User u = new User(email, name);

User saved = repo.save(u);

emailService.sendWelcomeEmail(saved);

return saved;

}

}

**Mockito Test**

// UserServiceTest.java

package com.example;

import static org.junit.jupiter.api.Assertions.\*;

import static org.mockito.Mockito.\*;

import org.junit.jupiter.api.Test;

import org.junit.jupiter.api.extension.ExtendWith;

import org.mockito.\*;

import org.mockito.junit.jupiter.MockitoExtension;

@ExtendWith(MockitoExtension.class)

class UserServiceTest {

@Mock

UserRepository repo;

@Mock

EmailService emailService;

@InjectMocks

UserService service;

@Captor

ArgumentCaptor<User> userCaptor;

@Test

void register\_ShouldSaveUser\_AndSendEmail() {

// Arrange

when(repo.save(any(User.class))).thenAnswer(inv -> {

User u = inv.getArgument(0);

u.setId(1L);

return u;

});

// Act

User saved = service.register("test@example.com", "John");

// Assert

assertEquals(1L, saved.getId());

assertEquals("John", saved.getName());

// Verify interactions

verify(repo, times(1)).save(any(User.class));

verify(emailService).sendWelcomeEmail(userCaptor.capture());

assertEquals("test@example.com", userCaptor.getValue().getEmail());

}

}

✅ **Benefits of this Example**

* **No real DB or email server** is used.
* The test runs in milliseconds.
* We focus only on **business logic**.
* Can simulate failures by doThrow(...).