**Junit Testing**

**@Test** annotation specifies that method is the test method.

**@Test(timeout=1000)** annotation specifies that method will be failed if it takes longer than 1000 milliseconds (1 second).

Test method should be public and void

**Methods annotated with the @Before annotation are run before each test.** This is useful when we want to execute some common code before running a test.

**@BeforeClass**: The @BeforeClass annotated method runs before the execution of test methods in a current class.

**@AfterClass**: The @AfterClass annotated method runs after the execution of test methods in a current class.

**Mockito**

**What is Mockito?**

Mockito is creating objects that simulate the behavior of real objects. They can be dynamically created from the code at runtime.

Offer more functionality than stubbing.

**What is Stubbing?**

Stubs are the objects that hold predefined data and uses it to give responses during tests. In other words, a stub is an object that resembles a real object with the minimum number of methods needed for a test. Stubs are used when we don't want to use objects that would give a response with real data.

**Spy Annotation**

A Spy in Mockito is a type of mock object that wraps an existing object. It allows you to call real methods on the object and still be able to verify that the method calls were made. A Spy can be useful when you want to test a method that calls other methods on the same object, but you don’t want to mock all the methods.

Allow us to call all the normal methods of the object while still tracking every interaction, just as we would with a mock.

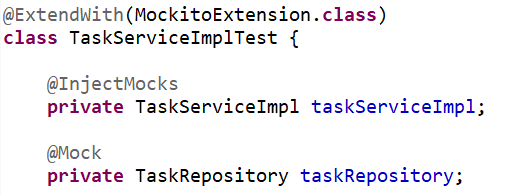
**@ExtendWith(MockitoExtension.class)**

Enables mockito annotations. Loads application context.

@Mock 🡪 To create mock objects

@InjectMock 🡪 create and inject the mock

e.g



**Service Layer Testing**

@ExtendWith(MockitoExtension.**class**)

**class** EmployeeServiceTest {

@Mock

EmployeeRepositoryImpl employeeRepository;

@Mock

JdbcTemplate jdbcTemplate;

@InjectMocks

EmployeeService employeeService;

**Get Mapping Testing for JDBC Template**

EmployeeService.java

**public** Employee getEmployeeById(Integer id) {

String sql = "SELECT \* FROM employee where id=?";

Employee employeeToReturn = **null**;

**try** {

employeeToReturn = jdbcTemplate.queryForObject(sql, **new** BeanPropertyRowMapper<>(Employee.**class**), id);

}

**catch**(DataAccessException e)

{

System.***out***.println(e.getMessage());

}

**return** employeeToReturn;

}

EmployeeServiceTest.java

@Test

@DisplayName("Get Employee By Id")

**public** **void** givenEmployeeId\_whenGetEmployee\_thenReturnEmployeeObj() {

//given

Address add1 = **new** Address("B/6 Shalimar", "Vakola Road", "Mumbai", "Maharashtra","India","123456");

Employee emp1 = **new** Employee("Mridul","Rajbhar",22,Role.***JR\_ENGINEER***,Gender.***MALE***, LocalDate.*of*(2000, 9, 12));

emp1.setAddress(add1);

*when*(jdbcTemplate.queryForObject(*anyString*(), *any*(BeanPropertyRowMapper.**class**), *anyInt*())).thenReturn(emp1);

//when

Employee employeeReturned = **this**.employeeService.getEmployeeById(1);

//then

*assertAll*(

()->{*assertNotNull*(employeeReturned);},

()->{*assertEquals*(emp1.getFirstname(), employeeReturned.getFirstname());}

);

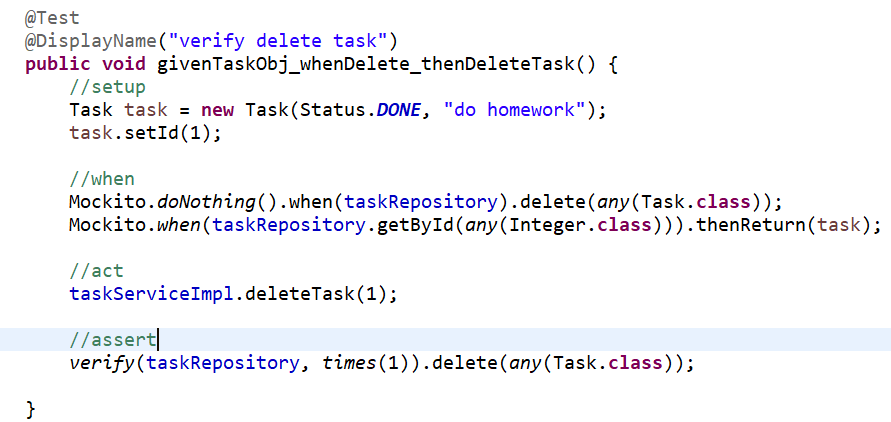
}

**GET Mapping Testing Hibernate**

**Exception Stubbing for void method**



**Delete Testing**

****

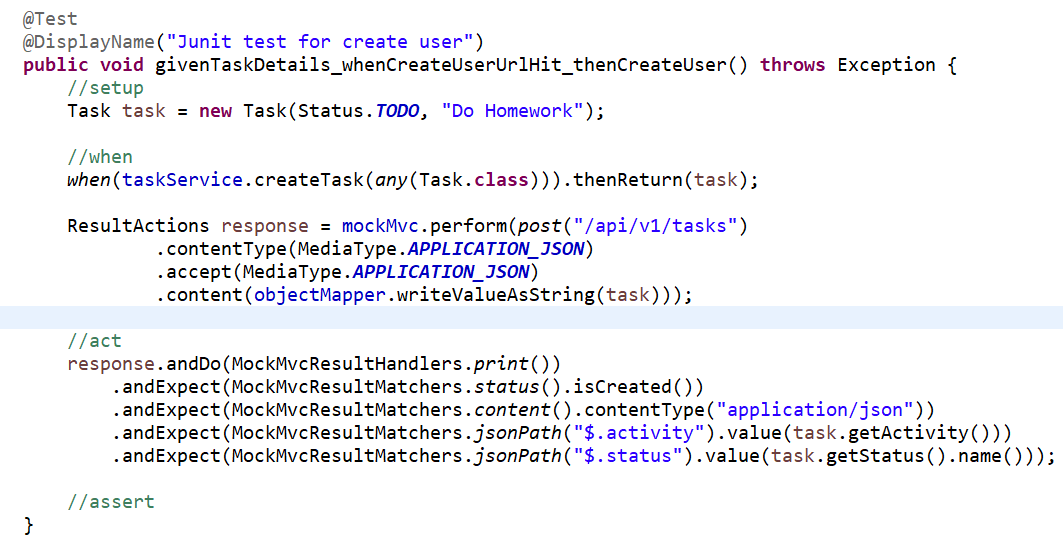
**Controller Layer Testing**

**@WebMvcTest 🡪** It will scan all the web layer(controller) class.

**@WebMvcTest(controller=TaskController.class) 🡪** It will scan only one mentioned controller.

@MockBean 🡪 Create mock object and put in spring application context.

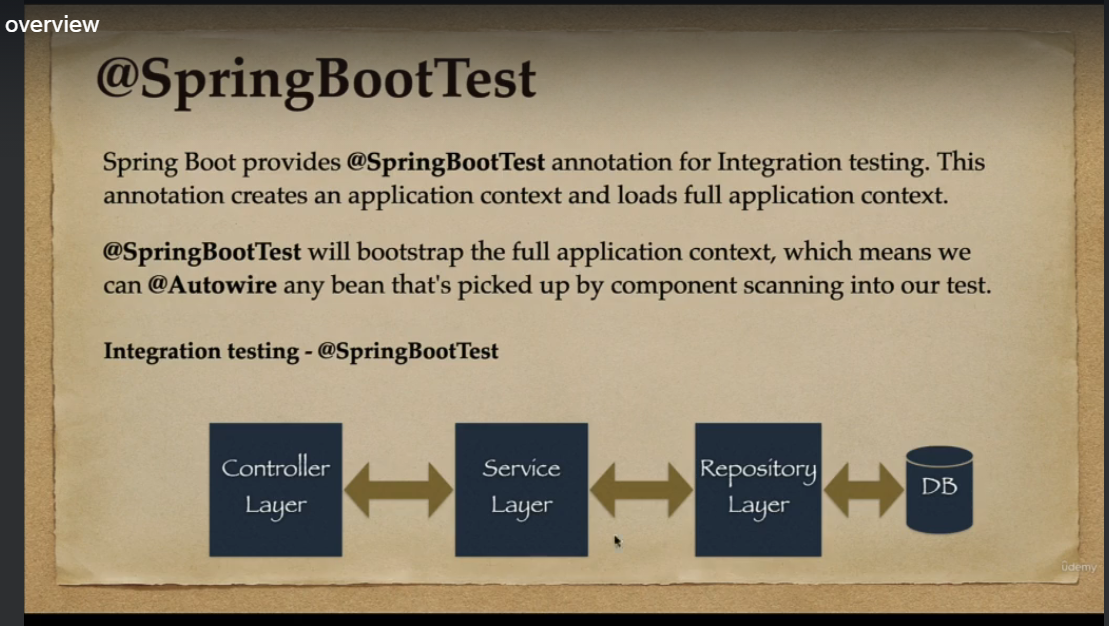
Create Operation



**Repository Layer Testing**

**@DataJpaTest -** It is used to test the repository layer and autoconfigures the emedded database for testing purpose.

**Integration Testing**



We don’t need to do mocking in Integration test