Junit:

- What is Junit and why we need Junit..?
- How to set up Junit..?
- What are all different types of annotations available in Junit..?
- How to write simple Junit test cases..?

Nagendrababu T

- To set up Junit in java project , follow these steps,
- Add Junit dependency in build management tools like maven or Gradle
- For Maven add below dependency

```
<dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
     <version>4.13.2</version>
    <scope>test</scope>
</dependency>
```

- For Gradle add below dependency

```
testImplementation 'junit:junit:4.13.2'
```

Nagendrababu T

- Types of Junit annotations
- @Test
- @Before
- @After
- @BeforeClass
- @AfterClass
- @BeforeEach
- @AfterEach
- @BeforeAll
- @AfterAll

Nagendrababu T

Below annotations serve same purpose for both Junit 4 and 5 versions

Junit 4

- @Before
- @BeforeClass
- @After
- @AfterClass

Junit5

- @BeforeEach
- @BeforeAll
- @AfterEach
- @AfterAll

Nagendrababu T

epam

@Before or @BeforeEach:

- It can be used to perform common setup tasks that need to be repeated before each test method.
- The method annotated with @Before or @BeforeEach is a non-static.

@BeforeClass or @BeforeAll:

- It can be used to perform common setup tasks that need to be executed only once per class execution
- The method annotated with @BeforeClass or @BeforeAll must be a static.

Nagendrababu T

epam

@After or @AfterEach:

- These annotations used to perform clean up or tear down tasks after each test method has been executed.
- The method annotated with @After or @AfterEach is a non-static.

@AfterClass or @AfterAll:

- These annotations used to perform clean up or tear down tasks after all test methods in a test class has been executed.
- The method annotated with @AfterClass or @AfterAll must be a static.

Nagendrababu T

Mockito

- Mockito is a popular mocking framework for Java that allows us to create and use mock objects in unit tests.
- We have different types of annotations present in Mockito
- **@Mock** this annotation used to create a mock object
- **@InjectMocks** this annotation used to automatically inject mock objects into the fields of the class under test
- **@RunWith(MockitoJunitRunner.class)** this annotation used to run the test class and initialize the mocks and handles the test execution

Nagendrababu T

How to set up Mockito

- Add below dependency in java project
- For Maven add below dependency

```
<dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.13.2</version>
    <scope>test</scope>
 </dependency>
```

- For Gradle add below dependency

```
testImplementation 'org.mockito:mockito-core:3.12.4'
```

Nagendrababu T

Power Mockito

- Power Mockito is an extension to the Mockito framework for testing and mocking complex code scenarios.

- It provides additional features and capabilities to mock static methods, private methods and final methods etc.

Nagendrababu T

Set up Power Mockito

- Add power Mockito dependency in java project
- For maven add below dependency ,
- **<dependency>**

```
        <groupId>org.powermock</groupId>
        <artifactId>powermock-api-mockito2</artifactId>
        <version>2.0.9</version>
        <scope>test</scope>
    </dependency>
```

Nagendrababu T

```
<dependency>
        <groupId>org.powermock</groupId>
        <artifactId>powermock-module-junit4</artifactId>
        <version>2.0.9</version>
        <scope>test</scope>
    </dependency>
```

For Gradle add below dependencies,

- **testImplementation 'org.powermock:powermock-core:2.0.9'**
- **testImplementation 'org.powermock:powermock-api-mockito2:2.0.9'**

Nagendrababu T

Important annotations present in Power Mockito**:**

- **@RunWith** annotation is used to specify the test runner to be used Power Mockito provides its own runner called **PowerMockRunner**

- **@PrepareForTest** used to specify the classes that needs to be prepared for testing including static, final, private methods

Nagendrababu T

epam

Summary:

- Choose **Junit** when there are no dependencies in a class and to test methods and verify the expected behavior and compare it with the actual output.
- Choose **Mockito** when you want to test a class that has dependencies on other classes or external resources. By creating mock objects for these dependencies, you can control their behavior and focus on testing.
- Choose **Power Mockito** when you want to test private, static, final methods. Internally it makes use of java Reflection.

Nagendrababu T

Thank you All !!!

Any Queries..?

Nagendrababu T

epam