JUnit 5 Basics and Differences from Previous Versions

Lesson Objectives

- Understanding the basic features of JUnit 5.
- Comparing JUnit 5 with its predecessors.
- Getting a glimpse of the JUnit 5 architecture.

Introduction to JUnit 5

JUnit 5 at a Glance

- Latest version of JUnit framework
- Launched in September 2017
- Comprises three sub-projects: JUnit Platform, JUnit Jupiter, and JUnit Vintage

JUnit 5 Architecture

- JUnit Platform:
  - Launches testing frameworks on the JVM and defines TestEngine API.
- JUnit Jupiter:
  - Provides a TestEngine for running Jupiter based tests.
- JUnit Vintage:
  - Provides TestEngine for running vintage tests, like JUnit 3 and 4.

JUnit 5 Basic Features

@Test Annotation

```java
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.assertEquals;

class CalculatorTest {

    @Test
    void addition() {
        Calculator calculator = new Calculator();
        assertEquals(5, calculator.add(2, 3));
    }
}
```

@BeforeEach and @AfterEach Annotations

```java
import org.junit.jupiter.api.AfterEach;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.assertNotNull;

class DatabaseTest {

    Database database;

    @BeforeEach
    void setUp() {
        database = new Database();
    }

    @Test
    void testConnection() {
        assertNotNull(database.getConnection());
    }

    @AfterEach
    void tearDown() {
        database.closeConnection();
    }

}
```

@BeforeAll and @AfterAll Annotations

```
import org.junit.jupiter.api.AfterAll;
import org.junit.jupiter.api.BeforeAll;
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.assertNotNull;

class ServerTest {

    static Server server;

    @BeforeAll
    static void init() {
        server = new Server();
        server.start();
    }

    @Test
    void testConnection() {
        assertNotNull(server.getConnection());
    }

    @AfterAll
    static void cleanup() {
        server.stop();
    }

}
```

Assertions in JUnit 5

```
// Asserts that the result of the addition is as expected.
assertEquals(5, calculator.add(2, 3), "2 + 3 should equal 5");

// Asserts that the email is valid.
assertTrue(validator.isValidEmail("test@example.com"), "Email should be valid");

// Asserts that the method does not return a null connection.
assertNotNull(database.getConnection(), "Connection should not be null");

// Groups assertions to check multiple properties of an object.
assertAll("person",
            () -> assertEquals("John", person.getFirstName(), "First name should be John"),
            () -> assertEquals("Doe", person.getLastName(), "Last name should be Doe")
        );
```

Grouping and Nesting Tests

```
import org.junit.jupiter.api.Nested;
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.assertTrue;

class NestedTestExample {

    @Nested
    class InnerClass {

        @Test
        void innerTest() {
            assertTrue(true);
        }
    }
}
```

@DisplayName and @Disabled Annotations

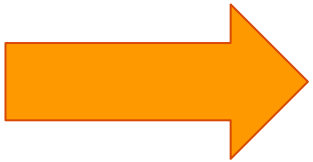
```java
import org.junit.jupiter.api.DisplayName;
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.assertTrue;

@DisplayName("A special container")
class DisplayNameExample {

    @Test
    @DisplayName("Custom test name containing spaces")
    void testWithDisplayNameContainingSpaces() {
        assertTrue(true);
    }
}
```

```java
import org.junit.jupiter.api.Disabled;
import org.junit.jupiter.api.Test;

class DisabledTestExample {

    @Test
    @Disabled("Disabled until bug #42 has been resolved")
    void testWillBeSkipped() {
    }
}
```

Differences Between JUnit 5 and Previous Versions

Annotation Comparison

```
@Before
@After
@BeforeClass
@AfterClass
```



```
@BeforeEach
@AfterEach
@BeforeAll
@AfterAll
```

Runners and Rules in JUnit 4

```java
// Custom Runner in JUnit 4
public class MyRunner extends BlockJUnit4ClassRunner
{

    public MyRunner(Class<?> klass) throws
InitializationError {
        super(klass);
    }

    @Override
    protected void runChild(FinalFrameworkMethod
method, RunNotifier notifier) {
        // Custom behavior
        super.runChild(method, notifier);
    }
}

// Usage of Custom Runner
@RunWith(MyRunner.class)
public class MyTests {
    @Test
    public void test() {
        // ...
    }
}
```

```java
// Custom Rule in JUnit 4
public class MyRule implements TestRule {

    public Statement apply(Statement base,
Description description) {
        return new Statement() {
            @Override
            public void evaluate() throws Throwable
{
                // Before
                base.evaluate();
                // After
            }
        };
    }
}

// Usage of Custom Rule
public class MyTests {
    @Rule
    public MyRule myRule = new MyRule();

    @Test
    public void test() {
        // ...
    }
}
```

Extension Model in JUnit 5

```java
// Custom Extension in JUnit 5
public class MyExtension implements
BeforeTestExecutionCallback,
AfterTestExecutionCallback {

    @Override
    public void
beforeTestExecution(ExtensionContext
context) throws Exception {
        // Before Test Execution
    }

    @Override
    public void
afterTestExecution(ExtensionContext
context) throws Exception {
        // After Test Execution
    }
}
```

```java
// Usage of Custom Extension
@ExtendWith(MyExtension.class)
public class MyTests {

    @Test
    public void test() {
        // ...
    }
}

// Combining Multiple Extensions in JUnit
5
@ExtendWith({MyExtension.class,
AnotherExtension.class})
public class MyTests {
    // ...
}
```

Constructor and Method Parameter Injection

```java
@ExtendWith(CustomResolver.class)
class ConstructorInjectionTest {

    private final String message;

    ConstructorInjectionTest(String message) {
        this.message = message;
    }

    @Test
    void testMessage() {
        System.out.println("Message: " + message);
    }
}
```

```java
@ExtendWith(CustomResolver.class)
class MethodInjectionTest {

    @Test
    void testMessage(String message) {
        System.out.println("Message: " + message);
    }
}
```

```java
class CustomResolver implements ParameterResolver {

    @Override
    public boolean supportsParameter(ParameterContext parameterContext, ExtensionContext extensionContext) throws
ParameterResolutionException {
        return parameterContext.getParameter().getType() == String.class;
    }

    @Override
    public Object resolveParameter(ParameterContext parameterContext, ExtensionContext extensionContext) throws
ParameterResolutionException {
        return "Injected Message";
    }
}
```

Tagging and Filtering

```java
import org.junit.jupiter.api.Tag;
import org.junit.jupiter.api.Test;

public class TaggingExample {

    @Test
    @Tag("fast")
    void fastTest() {
        // fast test code
    }

    @Test
    @Tag("slow")
    void slowTest() {
        // slow test code
    }
}
```

```java
import org.junit.jupiter.api.Tag;
import org.junit.jupiter.api.Test;
import org.junit.platform.runner.JUnitPlatform;
import org.junit.platform.suite.api.ExcludeTags;
import org.junit.platform.suite.api.IncludeTags;
import org.junit.platform.suite.api.SelectPackages;
import org.junit.runner.RunWith;

@RunWith(JUnitPlatform.class)
@SelectPackages("com.example")
@IncludeTags("fast")
@ExcludeTags("slow")
public class CustomTestSuite {

    // This class remains empty. It's used only as a holder for
the above annotations

}
```

Improved Exception Handling

```java
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.assertThrows;

public class ExceptionHandlingExample {

    @Test
    void exceptionTesting() {
        Exception exception = assertThrows(ArithmeticException.class, () ->
            { int result = 10 / 0; });  // This will throw ArithmeticException
    }
}
```

Transitioning to JUnit 5

Making the Move Towards Modern Testing

- Preparing for Transition
- Updating Dependencies
- Migrating Annotations
- Utilizing JUnit Vintage
- Overcoming Common Challenges