

# JUnit 5 Foundations

Test Framework



# JUnit 5 Foundations



- About JUnit 5
- Getting Start
- Contents of a Test Class
  - Lifecycle Methods
  - Test Methods
- Assertions & Assumptions
- Test Method Execution

# JUnit 5

A blue circle containing the text "JUnit Platform" in white.

JUnit Platform

A blue circle containing the text "JUnit Jupiter" in white.

JUnit Jupiter

A blue circle containing the text "JUnit Vintage" in white.

JUnit Vintage

- Java ပတ်ဝန်းကျင်မှာ Unit Test တွေကို ဆောင်ရွက်နိုင်ဖို့အတွက်ပြင်ဆင်ထားတဲ့ Test Framework တစ်ခုဖြစ်ပါတယ်
- JUnit 5 ကို အသုံးပြုနိုင်ဖို့အတွက် Java 8 နှင့်အထက်ကိုတော့ လိုအပ်ပါတယ်

# JUnit Platform

- Test Framework ကို JVM ပေါ်မှာ Launch လုပ်ဖို့အတွက် အခြေခံ Module တစ်ခုဖြစ်ပါတယ်
- Platform ပေါ်မှာ အလုပ်လုပ်ဖို့အတွက် TestEngine API ကို သတ်မှတ်ပေးပါတယ်
- Command Line ကနေ Run ဖို့အတွက် Console Launcher နဲ့ Custom Test Suite တွေကို Run ဖို့အတွက် JUnit Platform Suite Engine တို့ကိုလဲ Provide လုပ်ပေးထားပါတယ်
- Java IDEs တွေဖြစ်ကြတဲ့ IntelliJ Idea IDE, Eclipse, NetBeans နဲ့ VS Code တွေမှာလဲ Support လုပ်ပြီး၊ Build Tools တွေဖြစ်ကြတဲ့ Maven, Gradle, Ants Tools တွေမှာလဲ Support လုပ်ပါတယ်

# JUnit Jupiter

- JUnit 5 Test တွေကို ရေးသားဖို့အတွက် Programming Model နဲ့ Extension Model တွေကို Support လုပ်ထားပါတယ်
- Test Case တွေကို ရေးသားဖို့အတွက် Annotation တွေနဲ့ Extension တွေကို ရေးသားဖို့ Annotation တွေကို Support လုပ်ထားပါတယ်
- JUnit Jupiter Test တွေကို Run ဖို့အတွက် Test Engine ကိုလဲ Provide လုပ်ထားပါတယ်
- JUnit Jupiter ကို အသုံးပြုမယ်ဆိုရင် JUnit Jupiter API နဲ့ JUnit Jupiter Engine Maven Dependency တို့ကို Class Path ထဲမှာ ထည့်သွင်းထားဖို့လိုအပ်ပါတယ်

# JUnit Vintage

- JUnit 5 Platform ပေါ်မှာ JUnit 3, 4 Test Case တွေကို Run ဖို့အတွက် Test Engine ကို Support လုပ်ပါတယ်
- တကယ်လို့ JUnit Vintage ကို အသုံးပြုမယ်ဆိုရင် JUnit 4.12 နှင့်အထက်ကို Class Path ထဲမှာ ထည့်သွင်းထားဖို့လိုအပ်ပါတယ်
- အရင်ရေးထားခဲ့တဲ့ Test Case တွေကို JUnit 5 Platform ပေါ်မှာ အလုပ်လုပ်စေလိုတဲ့ အခါမှာ အသုံးပြုကြရမှာ ဖြစ်ပါတယ်

# Getting Start




JUnit Jupiter

Maven Dependencies

Writing Test Case

Running JUnit Test Case

# Maven Dependencies



```
<dependency>
  <groupId>org.junit.jupiter</groupId>
  <artifactId>junit-jupiter-api</artifactId>
  <version>5.8.2</version>
  <scope>test</scope>
</dependency>

<dependency>
  <groupId>org.junit.jupiter</groupId>
  <artifactId>junit-jupiter-engine</artifactId>
  <version>5.8.2</version>
  <scope>test</scope>
</dependency>
```



# JUnit Test Case

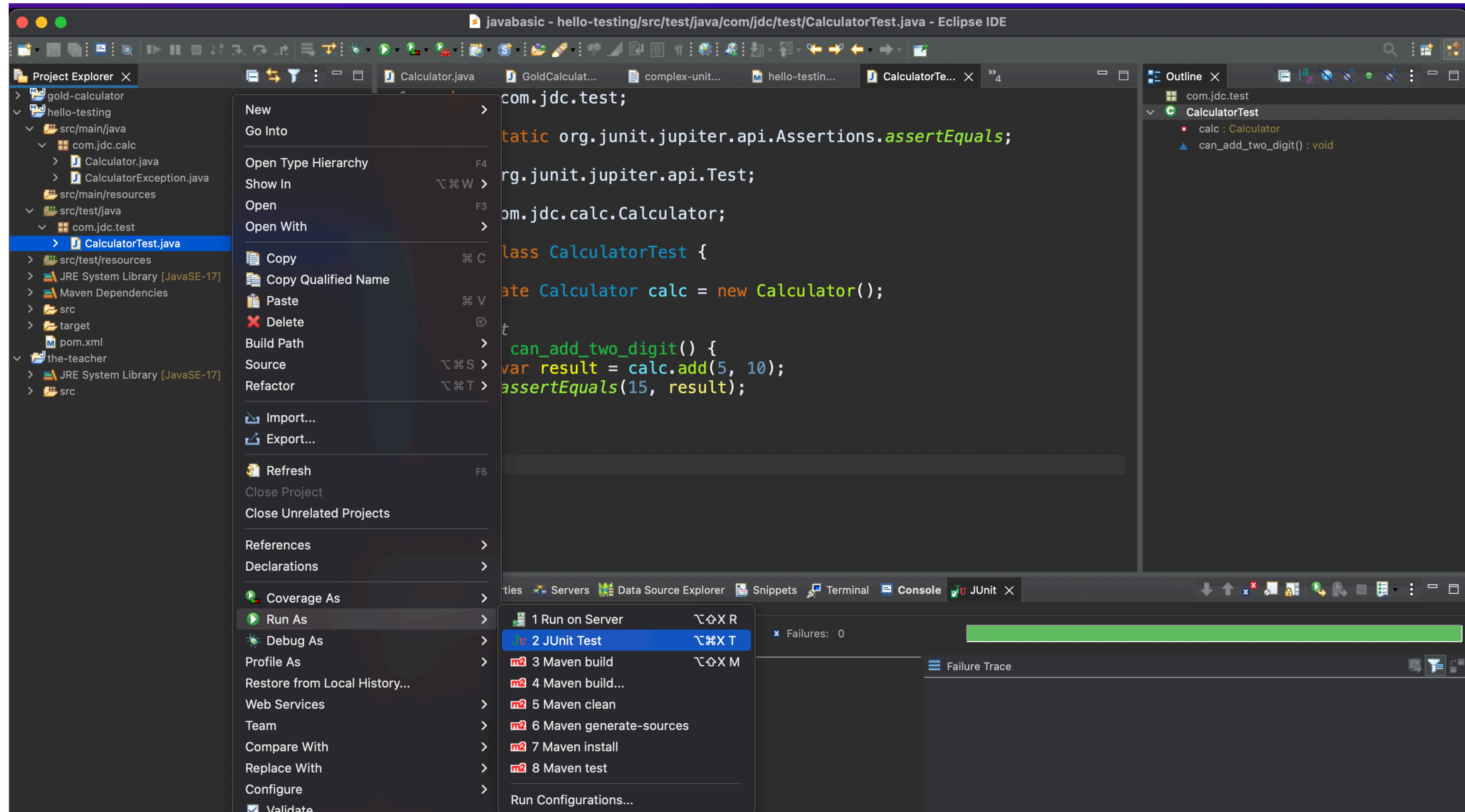
- Test Method တစ်ခုပါတဲ့ Top Level Class, Static Nested Class, Instance Class တွေကို JUnit Test Class အနေနဲ့ အသုံးပြုနိုင်ပါတယ်
- Instance Method တွေမှာ @Test Annotation ကို ရေးသားပြီး Test Case တစ်ခုအနေနဲ့ ရေးသားနိုင်ပါတယ်
- Instance Method ဖြစ်ပြီး၊ Void Method ဖြစ်မှသာ Test Case အနေနဲ့ အသုံးပြုနိုင်မှာ ဖြစ်ပါတယ်
- Parameterized Test တွေမှသာ Argument တွေကို ရေးသားအသုံးပြုနိုင်မှာ ဖြစ်ပြီး၊ ရိုးရိုး Test Case တွေဆိုရင်တော့ Argument တွေကို ရေးသားလို့ မရနိုင်ပါဘူး

# Writing Test Case



```
public class CalculatorTest {  
  
    private Calculator calc = new Calculator();  
  
    @Test  
    void can_add_two_digit() {  
        var result = calc.add(5, 10);  
        assertEquals(15, result);  
    }  
  
}
```

# Running JUnit Test Case



# Contents of a Test Class



JUnit Jupiter

Lifecycle Methods

Test Methods

# Lifecycle Methods

Annotation	Method	Description
@BeforeAll	Static	Test Class ထဲက Test Method တွေကို လုံးဝမစရသေးခင်မှာ တစ်ကြိမ် Invoke လုပ်ပေးမှာ ဖြစ်ပါတယ်
@BeforeEach	Instance	Test Class ထဲက Test Method တွေကို Invoke မလုပ်ခင်တိုင်းမှာ တစ်ကြိမ်စီ Invoke လုပ်ပေးမှာ ဖြစ်ပါတယ်
@AfterAll	Static	Test Class ထဲက Test Method တွေအားလုံးကို Invoke လုပ်ပြီးတဲ့ အခါမှာ တစ်ကြိမ် Invoke လုပ်ပေးမှာ ဖြစ်ပါတယ်
@AfterEach	Instance	Test Class ထဲက Test Method တွေကို Invoke လုပ်ပြီးတဲ့ အခါတိုင်းမှာ တစ်ကြိမ်စီ Invoke လုပ်ပေးမှာ ဖြစ်ပါတယ်

# Test Methods

Annotation	Description
<b>@Test</b>	Method တစ်ခုကို Test Method အဖြစ်သတ်မှတ်ပေးနိုင်ပါတယ်။
<b>@RepeatedTest</b>	Method တစ်ခုကို Repeated Test ရဲ့ Template Method အဖြစ်သတ်မှတ်ပေးနိုင်ပါတယ်။
<b>@ParameterizedTest</b>	Argument တွေပါတဲ့ Method တစ်ခုကို Test Method အဖြစ်သတ်မှတ်ပေးနိုင်ပါတယ်။
<b>@TestFactory</b>	Method တစ်ခုကို Dynamic Test တွေရဲ့ Method အဖြစ်သတ်မှတ်ပေးနိုင်ပါတယ်။
<b>@TestTemplate</b>	TestTemplateInvocationContextProvider နဲ့တွဲဖက်အသုံးပြုမည့် Test Template Method အဖြစ် သတ်မှတ်ပေးနိုင်ပါတယ်

# Display Names


- Test Class တွေ Test Method တွေမှာ Display Name ကို ပြောင်းပြီးပြလိုတဲ့ အခါမှာ @DisplayName Annotation ကို အသုံးပြုနိုင်ပါတယ်
- @DisplayName ရဲ့ Value Attribute တန်ဖိုးမှာ ပြလိုတဲ့ အမည်ကို ရေးသားပေးရပါမယ်
- Display Name တွေအတွက် Display Name Generator တွေကို ရေးသားပေးထားပြီး၊ လိုအပ်ပါက အသုံးပြုနိုင်မှာ ဖြစ်ပါတယ်

# Display Name Generators

Generator	Description
Standard	JUnit 5.0 မှာ စပြီးပါလာခဲ့တဲ့ Standard Display Name Generator ဖြစ်ပါတယ် Class ရဲ့ Simple Name ကို Test Class Name အဖြစ်အသုံးပြုပြီး၊ Method Name နေရာမှာ Method Name နဲ့ Argument List ကို ဖော်ပြပေးပါတယ်
Simple	Standard Generator ကို Extend လုပ်ထားပြီး Method Name နေရာမှာ Method Name နဲ့ Argument List ကို မပါပဲ ဖော်ပြပေးပါတယ်
ReplaceUnderscores	Simple Generator ကို Extends လုပ်ထားပြီး Method Name နေရာမှာ Underscore ကို Space နှင့် Replace လုပ်ပြီး ဖော်ပြပေးပါတယ်
IndicativeSentences	Test Class Display Name နှင့် Test Method Display Name တို့ကိုပေါင်းပြီး အဓိပ္ပါယ်ရှိသော ဝါကျတစ်ကြောင်းဖြစ်အောင် တည်ဆောက်ပေးနိုင်ပါတယ်။ JUnit 5.7 မှစပြီးပါလာတာ ဖြစ်ပါတယ်။



# Using Display Name Generator



```
@DisplayName("Calculator Test")
@DisplayNameGeneration(ReplaceUnderscores.class)
public class CalculatorTest {

    private Calculator calc = new Calculator();

    @Test
    void can_add_two_digit() {
        var result = calc.add(5, 10);
        assertEquals(15, result);
    }
}
```

can add two digit

# Indicative Sentences Generation



```
@DisplayName("Calculator Test")
@IndicativeSentencesGeneration(
    separator = " -> ",
    generator = ReplaceUnderscores.class
)
public class CalculatorTest {

    private Calculator calc = new Calculator();

    @Test
    void can_add_two_digit() {
        var result = calc.add(5, 10);
        assertEquals(15, result);
    }
}
```

Calculator Test -> can add two digit

# Assertions & Assumptions

- Test Method တွေကို ရေးသားတဲ့ နေရာမှာ Test လုပ်လိုတဲ့ Method Execution တွေ၊ Result တွေကို မှန်မမှန်စစ်တာတွေ၊ ပြီးတော့ Test Code တွေကို Execute လုပ်သင့်မလုပ်သင့်ဆိုတာတွေကို ရေးသားကြရမှာဖြစ်တယ်
- JUnit Framework မှာ Test Result တွေကို စစ်ဆေးဖို့အတွက် Assertion Method တွေကို Assertions Class ထဲမှာ Static Method တွေအနေနဲ့ ပုံပိုးပေးထားပါတယ်
- Test Code တွေကို ဆက်ပြီး Execute လုပ်သင့်မလုပ်သင့် ဆုံးဖြတ်နိုင်ရန်အတွက် Assumption Method တွေကို Assumptions Class ထဲမှာ Static Method တွေအနေနဲ့ ပုံပိုးပေးထားပါတယ်

# Assertions

## Standard Assertions

- Test Case တစ်ခုကို ရေးသားပြီးဆိုရင် Business Method တစ်ခုကို Invoke လုပ်ပြီး ရလာတဲ့ Result တွေကို ပြန်ပြီး စစ်ဆေးနိုင်ဖို့လိုအပ်ပါတယ်

## Group Assertions

- JUnit Jupiter မှာ JUnit 4 မှာကထက် အသုံးပြုခဲ့တဲ့ Assertion Method တွေကို အသုံးပြုနိုင်သလို၊ Java 8 ရဲ့ Lambda Style နဲ့ ရေးသားနိုင်တဲ့ Method တို့ကိုလဲ ဖြည့်စွက်ထားပါတယ်

## Dependent Assertions

## Assertions for Exceptions

- JUnit Jupiter မှာ JUnit ရဲ့ Built In Assertion Method တွေကိုအသုံးပြုပြီး Assertions တွေကို ရေးသားနိုင်သလို၊ လိုအပ်ပါက Third Party Assertion Framework တွေနဲ့လဲ တွဲဖက်ရေးသားနိုင်ပါတယ်

## Time Out Assertions

# Standard Assertions

Assertion Methods		Description
assertNull	assertNotNull	Null တန်ဖိုးဟုတ်မဟုတ် စစ်ပေးနိုင်
assertTrue	assertFalse	True လား False လား ဆိုတာကို စစ်ပေးနိုင်
assertSame	assertNotSame	Object နှစ်ခုဟာ တစ်ခုထဲသော Instance ဟုတ်မဟုတ် စစ်ပေးနိုင်
assertEquals	assertNotEquals	တန်ဖိုးနှစ်ခု တူမတူဆိုတာကို စစ်ပေးနိုင်
assertArrayEquals	-	Array နှစ်ခုရဲ့ တန်ဖိုးတွေ တူမတူဆိုတာကို စစ်ပေးနိုင်
assertIterableEquals	-	Collection နှစ်ခုတူမတူဆိုတာကို စစ်ပေးနိုင်
assertLinesMatch	-	String List Object နှစ်ခုရဲ့ တန်ဖိုးတွေတူမတူဆိုတာကို စစ်ပေးနိုင်

# Group Assertions

- Assertion တွေကို စုစည်းပြီး ရေးသားနိုင်ဖို့အတွက် `assertAll()` Method ကို ပြင်ပေးထားပါတယ်။
- `assertAll()` Method တွေမှာ အခြေခံအားဖြင့် `Executable Interface Object` ကို `Varargs` ပုံစံဖြင့် ရေးသားနိုင်သလို၊ `Collection` ပုံစံအနေနဲ့လဲ ရေးသားနိုင်သလို၊ `Stream` အနေနဲ့လဲ ရေးသားနိုင်ပါတယ်
- `Executable Interface` ဟာ `Functional Interface` ဖြစ်တဲ့အတွက် `Java 8` ရဲ့ `Lambda Style` ဖြင့်လဲ ရေးသားနိုင်မှာ ဖြစ်ပါတယ်

# Group Assertions via Varargs

```
public class GroupTesting {  
  
    Member member;  
  
    @BeforeEach  
    void before() {  
        member = new Member("Aung Aung", "Admin");  
    }  
  
    @Test  
    void test_with_varargs() {  
  
        assertAll("Member Test",  
            () -> assertNotNull(member),  
            () -> assertEquals(member.getName(), "Aung Aung"),  
            () -> assertEquals(member.getRole(), "Admin")  
        );  
    }  
}
```



# Group Assertions via Collection

```
public class GroupTesting {  
  
    Member member;  
  
    @BeforeEach  
    void before() {  
        member = new Member("Aung Aung", "Admin");  
    }  
  
    @Test  
    void test_with_list() {  
  
        assertAll("Member Test",  
            List.of(  
                () -> assertNotNull(member),  
                () -> assertEquals(member.getName(), "Aung Aung"),  
                () -> assertEquals(member.getRole(), "Admin")  
            )  
        );  
    }  
}
```



# Group Assertions via Stream

```
public class GroupTesting {

    Member member;

    @BeforeEach
    void before() {
        member = new Member("Aung Aung", "Admin");
    }

    @Test
    void test_with_stream() {

        assertAll("Member Test",
            Stream.of(
                () -> assertNotNull(member),
                () -> assertEquals(member.getName(), "Aung Aung"),
                () -> assertEquals(member.getRole(), "Admin")
            )
        );
    }
}
```

# Dependent Assertions

```
@Test
void test_with_dependents() {

    assertAll("Member",
        () -> {
            // Check Member not null
            assertNotNull(member);

            assertAll("Name", () -> {
                // Check Name
                assertNotNull(member.getName());

                assertAll(
                    () -> assertTrue(member.getName().startsWith("A")),
                    () -> assertTrue(member.getName().endsWith("g"))
                );
            });

            assertEquals(member.getRole(), "Admin");
        }
    );
}
```

# Assertion for Exceptions

- Test Case တွေကို ရေးသားတဲ့အခါမှာ မှန်တဲ့ Test တွေကိုလဲ ရေးသားဖို့လိုအပ်သလို၊ Exception ဖြစ်နိုင်တဲ့ အနေအထားအတွက် Test တွေကိုလဲ ရေးသားဖို့ လိုအပ်ပါတယ်
- JUnit 5 မှာ Exception တွေကို စစ်ဆေးနိုင်ဖို့ Assertions Method တွေကို ပြင်ပေးထားပါတယ်
- `assertThrow()` Method ဖြင့် ဖြစ်နိုင်တဲ့ Exception Object တွေကို စစ်ဆေးနိုင်သလို၊ Exception ရဲ့ State တွေကိုလဲ စစ်ဆေးနိုင်မှာ ဖြစ်ပါတယ်

# Assertion for Exceptions

```
@Test
void test_for_exceptions() {

    var exception = assertThrows(
        // Expected Exception Class
        ArithmeticException.class,
        () -> {
            // Code Execution
            System.out.println(10 / 0);
        }
    );

    // Check Error Message of Exception Object
    assertEquals("/ by zero", exception.getMessage());
}
```

# Time Out Assertions

```
@Test
void test_for_time_out_execution() {

    // Assertion will fails when execution exceed 3 Seconds
    assertTimeout(Duration.ofSeconds(3), () -> {

        // Long Task
        try {
            // Will Fail
            Thread.sleep(3500);
        } catch (Exception e) {
            e.printStackTrace();
        }
    });
}
```

# Assumption

- Assertions Method တွေကတော့ Result တွေကို မှန်မမှန် စစ်ပေးနိုင်တာဖြစ်ကြပြီး၊ Assumption Method တွေကတော့ Condition အပေါ်မူတည်ပြီး Test ကို Execution ဆက်လုပ်သင့်မလုပ်သင့် ဆုံးဖြတ်နိုင်ကြတဲ့ Method တွေဖြစ်ကြပါတယ်
- JUnit 4 မှာကထဲက အသုံးပြုခဲ့တဲ့ `assumeTrue()`, `assumeFalse()` Methods တွေအပြင် Lambda Style နဲ့ ရေးသားနိုင်တဲ့ Assumption Methods တွေကို ပြင်ဆင်ပေးထားပါတယ်
- Assumption Result ဟာ Fails ဖြစ်ခဲ့ရင် Error မဟုတ်ပဲ Test Method ကို Ignore လုပ်ပေးသွား မှာ ဖြစ်ပါတယ်

# Legacy Assumption




```
@Test
void assumption_legacy() {

    // Assumption
    assertTrue("DEV".equals(System.getenv("profile")));

    // remainder of test
    assertTimeout(Duration.ofSeconds(3), () -> {
        // Long Task
        try {
            Thread.sleep(2900);
        } catch (Exception e) {
            e.printStackTrace();
        }
    });
}
```

# Lambda Style Assumption



```
@Test
void assumption_lambda() {
    assumingThat(
        // Assumption
        "DEV".equals(System.getenv("profile")),
        // When Success
        () -> {
            // remainder of test
        });
}
```



# Test Method Execution

- JUnit Jupiter မှာ Test Method တွေရဲ့ Execution တွေကို Annotation တွေနဲ့ Handle လုပ်နိုင်အောင်ပြင်ဆင်ပေးထားပါတယ်
- @Disabled Annotation ကို အသုံးပြုပြီး အကြောင်းတစ်ခုခုကြောင့် Execute မလုပ်လိုသေးတဲ့ Test Method ကို တားထားနိုင်ပါတယ်
- ထို့အပြင် Condition တွေကို အသုံးပြုနိုင်တဲ့ Annotation တွေကို ပြင်ပေးထားပြီး Condition နဲ့ ကိုက်ညီရင် Enabled လုပ်မယ်၊ Disabled လုပ်မယ် ဆိုတာကိုလဲ Handle လုပ်နိုင်အောင် ပြင်ပေးထားပါတယ်

# Conditional Annotations

Annotations		Description
@DisabledIf	@EnabledIf	Condition အပေါ်မူတည်ပြီးဆုံးဖြတ်
@DisabledIfEnvironmentVariable	@EnabledIfEnvironmentVariable	Environment Variable အပေါ်မူတည်ပြီးဆုံးဖြတ်
@DisabledIfSystemProperty	@EnabledIfSystemProperty	System Property အပေါ်မူတည်ပြီးဆုံးဖြတ်
@DisabledOnJre	@EnabledOnJre	JRE Version အပေါ်မူတည်ပြီး ဆုံးဖြတ်
@DisabledOnOs	@EnabledOnOs	OS အပေါ်မူတည်ပြီး ဆုံးဖြတ်