
# Junit 5

Creating Tests - Features

# Meta-Annotation

```java
@Target(ElementType.METHOD)
@Retention(RetentionPolicy.RUNTIME)
@Tag("fast")
@Test
public @interface FastTest {
}
```

---

```java
@FastTest
public void testFast() {
    System.out.println("This is a fast test");
}
```

```
mvn clean test -Dgroups="fast"
[INFO] Results:
[INFO]
[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0
[INFO]
```

# Assumptions

```
@Test
void testOnlyOnCiServer() {
    assumeTrue("CI".equals(System.getProperty("ENV")),
        "Aborting test: not on CI server");
    // The rest of the test code goes here
}
```

```
mvn test -Dtest=AssumptionsDemo#testOnlyOnCiServer
```

```
[INFO] Results:
[INFO]
[WARNING] Tests run: 1, Failures: 0, Errors: 0, Skipped: 1
```

```
mvn test -Dtest=AssumptionsDemo#testOnlyOnCiServer -DENV=CI
```

```
[INFO] Results:
[INFO]
[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0
```

# Assumptions - Test on all environments

```
@Test
void testInAllEnvironments() {
    assumingThat("CI".equals(System.getenv("ENV")),
        () -> {
            // perform these assertions only on the CI server
            assertEquals(2, calculator.divide(4, 2));
        });

    // perform these assertions in all environments
    assertEquals(42, calculator.multiply(6, 7));
}
```

# @Disabled

```
@Disabled("Disabled until bug #99 has been fixed")
public class DisabledClassDemo {

    @Test
    void testWillBeSkipped() {
    }
}
```

```
public class DisabledTestsDemo {

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

    @Test
    void testWillBeExecuted() {
    }
}
```

# Conditional Tests - OS

```java
@Test
@EnabledOnOs(WINDOWS)
void onlyOnWindowsOs() {
    System.out.println("This test will only run on Windows");
}
```

```java
@Test
@EnabledOnOs({ LINUX, MAC })
void onLinuxOrMac() {
    System.out.println("This test will run on Linux or macOS");
}
```

```java
@Test
@DisabledOnOs(WINDOWS)
void notOnWindows() {
    System.out.println("This test will not run on Windows");
}
```

# Conditional Tests - Architecture

```java
@Test
@DisabledOnOs(architectures = "x86_64")
void notOnX86_64() {
    System.out.println("This test will not run on x86_64");
}
```

```java
@Test
@EnabledOnOs(value = MAC, architectures = "aarch64")
void onNewMacs() {
    System.out.println("This test will run on new Macs");
}
```

```java
@Test
@DisabledOnOs(value = MAC, architectures = "aarch64")
void notOnNewMacs() {
    System.out.println("This test will not run on new Macs");
}
```

# Conditional Tests - System and Environment Properties

```java
@Test
@EnabledIfSystemProperty(named = "os.arch", matches = ".*64.*")
void onlyOn64BitArchitectures() {
    System.out.println("This test will only run on 64-bit architectures");
}
```

```java
@Test
@DisabledIfSystemProperty(named = "ci-server", matches = "true")
void notOnCiServer() {
    System.out.println("This test will not run on the CI server");
}
```

```java
@Test
@EnabledIfEnvironmentVariable(named = "ENV", matches = "staging-server")
void onlyOnStagingServer() {
    System.out.println("This test will only run on the staging server");
}
```

# Conditional Tests - Custom Condition

```
@Test
@EnabledIf("customCondition")
void enabled() {
    System.out.println("This test is custom condition - enabled");
}

@Test
@DisabledIf("customCondition")
void disabled() {
    System.out.println("This test is custom condition - disabled");
}

boolean customCondition() {
    return true;
}
```

# Conditional Tests - Custom Condition

```java
package example;

import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.condition.EnabledIf;

class ExternalCustomConditionDemo {

    @Test
    @EnabledIf("example.ExternalCondition#customCondition")
    void enabled() {
        // ...
    }
}
```

```java
class ExternalCondition {

    static boolean customCondition() {
        return true;
    }
}
```

# Nested Tests

- Possible
- Not recommended
- Look it up in the user guide

# Native Junit Dependency Injection

# What is Dependency Injection?

# Native Junit 5 DP Objects

- Native Junit 5 dependency injection
  - TestInfo object
  - RepetitionInfo object
  - TestReporter object
  - TBD
    - ExtensionContext object
    - Extension model

# Native Junit 5 DI Objects

```
@Tag("TestInfoTest")
@DisplayName("Test info DI Test")
@Test
public void TestInfoDITest(TestInfo info) {
    System.out.println("Tags: " + info.getTags());
    System.out.println("Test class: " + info.getTestClass());
    System.out.println("Test method: " + info.getTestMethod());
}
```

```
@RepeatedTest(3)
public void RepetitionDITest(RepetitionInfo info) {
    System.out.println("Current repetition: " +
info.getCurrentRepetition());
    System.out.println("Total repetitions: " + info.getTotalRepetitions());
    System.out.println("Failure count: " + info.getFailureCount());
}
```

# Native Junit 5 DI Objects

```
@AfterEach
public void afterEach(TestReporter reporter) {
    reporter.publishEntry("after", "report");
}
```

```
@RepeatedTest(3)
public void RepetitionDITest(RepetitionInfo info) {
    System.out.println("Current repetition: " +
info.getCurrentRepetition());
    System.out.println("Total repetitions: " + info.getTotalRepetitions());
    System.out.println("Failure count: " + info.getFailureCount());
}
```

# Parameterized Tests

# Intro to Parameters

- Run multiple times with different arguments
- Consumes arguments via hard coded / Enum / file / method arguments
- Resolve parameter into the test method
- Can resolve any type of arguments
- When should we use it?

# @ValueSource

```
@ParameterizedTest
@ValueSource(ints = { 1, 2, 3 })
void palindromes(int candidate) {
    Assertions.assertEquals(candidate, 2);
}
```

```
@ParameterizedTest
@ValueSource(strings = { "alpha", "beta", "charly" })
void palindromes(String candidate) {
    Assertions.assertEquals(candidate.length(), 5);
}
```

# @EnumSource

```
@ParameterizedTest
@EnumSource(ChronoUnit.class)
void testWithEnumSource(TemporalUnit unit) {
    assertNotNull(unit);
}
```

```
@ParameterizedTest
@NullSouce
@EmptySource
@EnumSource(ChronoUnit.class)
void testWithEnumSource(TemporalUnit unit) {
    assertNotNull(unit);
}
```

There’s more on the subject. Visit the manual

# @MethodSource

```
@ParameterizedTest
@MethodSource("stringProvider")
void testWithExplicitLocalMethodSource(String argument) {
    assertNotNull(argument);
}

static Stream<String> stringProvider() {
    return Stream.of("apple", "banana");
}
```

# @MethodSource

```java
@ParameterizedTest
@MethodSource("range")
void testWithRangeMethodSource(int argument) {
    assertNotEquals(9, argument);
}

static IntStream range() {
    return IntStream.range(0, 20).skip(10);
}
```

# @MethodSource

```
@ParameterizedTest
@MethodSource("stringIntAndListProvider")
void testWithMultiArgMethodSource(String str, int num, List<String> list) {
    assertEquals(5, str.length());
    assertTrue(num >=1 && num <=2);
    assertEquals(2, list.size());
}

static Stream<Arguments> stringIntAndListProvider() {
    return Stream.of(
            arguments("apple", 1, Arrays.asList("a", "b")),
            arguments("lemon", 2, Arrays.asList("x", "y"))
    );
}
```

Method can be also external to the test class. Visit the manual

# @CSVSource

```
@ParameterizedTest
@CsvSource({
        "apple,         1",
        "banana,        2",
        "'lemon, lime', 0xF1",
        "strawberry,    700_000"
})
void testWithCsvSource(String fruit, int rank) {
    assertNotNull(fruit);
    assertNotEquals(0, rank);
}
```

# @CSVSource

```
@ParameterizedTest
@CsvFileSource(resources = "/two-column.csv", numLinesToSkip = 1)
void testWithCsvFileSourceFromClasspath(String country, int reference) {
    assertNotNull(country);
    assertNotEquals(0, reference);
}
```

*two-column.csv*

```
COUNTRY, REFERENCE
Sweden, 1
Poland, 2
"United States of America", 3
France, 700_000
```

Fine tuning exists. Visit the manual

# @ArgumentSource

```
@ParameterizedTest
@ArgumentsSource(MyArgumentsProvider.class)
void testWithArgumentsSource(String argument) {
    assertNotNull(argument);
}
```

```
public class MyArgumentsProvider implements ArgumentsProvider {

    @Override
    public Stream<? extends Arguments> provideArguments(ExtensionContext context) {
        return Stream.of("apple", "banana").map(Arguments::of);
    }
}
```

More on extensions - later

# Argument Aggregator

```
@ParameterizedTest
@CsvSource({
    "Jane, Doe, F, 1990-05-20",
    "John, Doe, M, 1990-10-22"
})
void testWithArgumentsAccessor(ArgumentsAccessor arguments) {
    Person person = new Person(arguments.getString(0),
                               arguments.getString(1),
                               arguments.get(2, Gender.class),
                               arguments.get(3, LocalDate.class));

    if (person.getFirstName().equals("Jane")) {
        assertEquals(Gender.F, person.getGender());
    }
    else {
        assertEquals(Gender.M, person.getGender());
    }
    assertEquals("Doe", person.getLastName());
    assertEquals(1990, person.getDateOfBirth().getYear());
}
```

# Interface and Default Methods

```
@TestInstance(Lifecycle.PER_CLASS)
interface TestLifecycleLogger {

    static final Logger logger = Logger.getLogger(TestLifecycleLogger.class.getName());

    @BeforeEach
    default void beforeEachTest(TestInfo testInfo) {
        logger.info(() -> String.format("About to execute [%s]",
            testInfo.getDisplayName()));
    }

    @AfterEach
    default void afterEachTest(TestInfo testInfo) {
        logger.info(() -> String.format("Finished executing [%s]",
            testInfo.getDisplayName()));
    }

}
```

# Interface and Default Methods

```
class TestInterfaceDemo implements TestLifecycleLogger, AnotherExtension {

    @Test
    void isEqualValue() {
        assertEquals(1, "a".length(), "is always equal");
    }
}
```


# To be continued…