Embracing JUnit 5 with eclipse

Noopur Gupta
Eclipse JDT UI co-lead
IBM India
noopur_gupta@in.ibm.com
@noopur2507
JUnit Framework

- **JUnit 4.0**
  Released in 2006

- **JUnit 5.0**
  Release in Q3 2017 (expected)

---

### The Top 20 Libraries Used by Github's Most Popular Java Projects

- junit
- sf4j-api
- guava
- log4j
- commons-io
- sf4j-log4j12
- mockito-all
- commons-lang
- logback-classic
- commons-lang3
- servlet-api
- apache-httpclient
- spring-context
- jackson-databind
- commons-codec
- mockito-core
- spring-test
- joda-time
- google-gson
- testing

JUnit 4 Architecture

Problems

- Modularity (single JAR)
  - Developers accessing internals and duplicating code
  - Maintainability

- Extensibility
  - (via Runners and Rules)

- Java 8 support
  - (Lambdas, Streams, Default Methods etc.)

@test developers

IDEs developers

Build tools developers

Testing frameworks developers

junit.jar
(Single JUnit 4 JAR)
(+ org.hamcrest.core.jar)
JUnit 5 Architecture

JUnit 5 = Platform + Jupiter + Vintage

JUnit 5 Architecture

JUnit Platform
- junit-platform-launcher
  Launcher APIs used by IDEs and build tools to launch the framework.
  Finds test engines via Java’s ServiceLoader mechanism.
- junit-platform-engine
  TestEngine APIs for integration of any testing framework like JupiterTestEngine.
  ...

JUnit Jupiter
- junit-jupiter-api
  APIs for the new programming and extension model.
- junit-jupiter-engine
  To discover and execute Jupiter tests.

JUnit Vintage
- junit-vintage-engine
  To discover and execute JUnit 3 and JUnit 4 tests on JUnit 5 platform.
A sneak peek into the major interesting features of JUnit Jupiter with Eclipse beta support for JUnit 5.
Eclipse beta support for JUnit 5

Follow the steps provided in the [wiki](#) to setup JUnit 5 support in Eclipse.

Demo: Eclipse 4.7 M6 build with JUnit 5 support based on JUnit 5 M3 release (using JUnit5-20170313-snapshot JARs).
Demo

JUnit Jupiter - Programming Model
Create new JUnit Jupiter test in Eclipse with setUp and tearDown methods for a class and its methods being tested:

New JUnit Test Case wizard > New JUnit Jupiter test
Demo

- Add JUnit 5 library to the build path

  ➢ New JUnit Test Case wizard offers to add it while creating a new JUnit Jupiter test.

  ➢ Quick Fix (Ctrl+1) proposal on @Test.
Demo

- Visibility: Test classes and methods can have any access modifier (other than private).

- Annotations

<table>
<thead>
<tr>
<th>JUnit 4</th>
<th>JUnit 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>@org.junit.Test</td>
<td>@org.junit.jupiter.api.Test (No expected and timeout attributes)</td>
</tr>
<tr>
<td>@BeforeClass</td>
<td>@BeforeAll</td>
</tr>
<tr>
<td>@AfterClass</td>
<td>@AfterAll</td>
</tr>
<tr>
<td>@Before</td>
<td>@BeforeEach</td>
</tr>
<tr>
<td>@After</td>
<td>@AfterEach</td>
</tr>
<tr>
<td>@Ignore</td>
<td>@Disabled</td>
</tr>
</tbody>
</table>
Demo

Create a JUnit Jupiter test method in Eclipse with the new template:
Demo - Assertions

-org.junit.jupiter.api.Assertions class-

- Failure message comes at the end of arguments list.

- Failure message can be retrieved lazily.

- Grouped assertions to execute all assertions first and then report all failures together.

- Exception testing to assert and evaluate a thrown exception.

- Asserting that the given task completes before the given timeout.
Demo - Assumptions

(org.junit.jupiter.api.Assumptions class)

```java
@Test
void testOnlyOnMac() throws Exception {
    assertTrue("macOS".equals(System.getProperty("os.name")), () -> {
        return "aborting test: requires macOS, found " + System.getProperty("os.name");
    });

    // test code for macOS
    assertTrue(new File("/Volumes/").exists());
}

@Test
void testOnAllSystems() throws Exception {
    assumingThat("macOS".equals(System.getProperty("os.name")), () -> {
        assertTrue(new File("/Volumes/").exists());
    });

    // test code for all systems
    assertTrue(1 + 1 == 2);
}
```
JUnit Jupiter’s *Assertions, Assumptions & DynamicTest* classes are now added to Eclipse Favorites by default.

Import static methods in your code from favorite classes via Content Assist (*Ctrl + Space*) and Quick Fix (*Ctrl + 1*).

Configure the number of static member imports needed before type.* is used.
Demo - @DisplayName

Provide custom display names for test classes and test methods - with spaces, special characters, and even emojis!

Use "Go to File" action or just double-click to navigate to the test from JUnit view.
Demo - @Nested Test Classes

Non-static nested classes (i.e. inner classes) can serve as @Nested tests for logical grouping of test cases.

Example: TestingAStackDemo in JUnit 5 user guide.
Demo - Interface Default Methods

- Default test methods can be inherited by implementing test classes.
- Enables multiple inheritance in test classes.

Example: StringTests in JUnit 5 user guide.

default void valueEqualsItself() {
    T value = createValue();
    assertEquals(value, value);
}

class StringTests implements ComparableContract<String>, EqualsContract<String> {
    @Override
    public String createValue() {
        return "foo";
    }
    @Override
    public String createSmallerValue() {
        return "bar"; // 'b' < 'f' in "foo"
    }
    @Override
    public String createNotEqualValue() {
        return "baz";
    }
}
Demo - Tagging, Meta-Annotations and Composed Annotations

• Tag test classes and test methods with @Tag. Tags can later be used to filter test execution.

```java
@Tag("performance")
@Test
void testPerformance() {
}
```

• JUnit Jupiter annotations can be used as meta-annotations.

• Create custom composed annotation inheriting semantics of its meta-annotations.

```java
@Target({ ElementType.TYPE, ElementType.METHOD })
@Retention(RetentionPolicy.RUNTIME)
@Tag("performance")
@Test
@interface PerformanceTest {
}

void testPerformance() {
}
```
Demo - Dynamic Tests

- Dynamic tests are generated at runtime by a `@TestFactory` method.

Create a `@TestFactory` method in Eclipse with the new template:

```java
class DynamicTestsDemo {
    @TestFactory
    Stream<DynamicTest> testFactoryName() throws Exception {
        // TODO: generate dynamic test cases
        return null;
    }
}
```

- Dynamic test is composed of a display name and an Executable.

```java
@TestFactory
Stream<DynamicTest> dynamicTests() throws Exception {
    return Stream.of(1, 2).map(
        i -> dynamicTest("dynamic test for: " + i, () -> isEven(i)));
}
```
Demo - Navigate to failing source location

Double-click an entry in JUnit view’s failure trace to jump to the corresponding source location.

Click "Show Stack Trace in Console View" and use the hyperlinks. It can also be used to copy parts of the stack trace.
Demo – Dependency injection

• Test constructors and methods are now permitted to have parameters enabling Dependency Injection.

• `ParameterResolver` is the extension API to provide a parameter resolver which dynamically resolves a parameter at runtime.

• Two built-in resolvers by JUnit Jupiter that are registered automatically:
  - `TestInfoParameterResolver` for parameter type `TestInfo`
  - `TestReporterParameterResolver` for parameter type `TestReporter`
JUnit Jupiter - Extension Model

- Provides extension points as interfaces in `org.junit.jupiter.api.extension` package to be implemented by extension providers.

- Register one or more extensions on a test class, test method, or composed annotation with `@ExtendWith(...)`.  

```
@ExtendWith({ FooExtension.class, BarExtension.class })
class MyTest {
  // ...
}
```

See the JUnit 5 User Guide for more details.
Resources

➢ JUnit 5 Project
   http://junit.org/junit5
   ▪ User Guide
     http://junit.org/junit5/docs/current/user-guide
   ▪ Javadoc
     http://junit.org/junit5/docs/current/api
   ▪ GitHub
     https://github.com/junit-team/junit5
   ▪ Q&A
     http://stackoverflow.com/questions/tagged/junit5

➢ Eclipse support for JUnit 5
   https://wiki.eclipse.org/JDT UI/JUnit_5
Evaluate the Sessions

Sign in and vote at eclipseconverge.org

-1  0  +1
Thank You!

That's all Folks!