

# Leveraging Test Lifecycle

---

## Understanding Junit Test Lifecycle



**Jim Weaver**

Developer, Trainer and Author

[www.codeweaver.org](http://www.codeweaver.org)



```
class DurationUnitTest {  
  
    @Test  
    public void matchUnitBySingularString() {  
        assertSame(DurationUnit.WEEK, DurationUnit.getByTextValue("week"));  
    }  
  
    @Test  
    public void matchUnitByPluralString() {  
        assertSame(DurationUnit.WEEK, DurationUnit.getByTextValue("weeks"));  
    }  
  
    @Test  
    public void returnsNullForUnmatchedUnit() {  
        assertNull(DurationUnit.getByTextValue("boop"));  
    }  
}
```

## How are test classes instantiated when tests are run?

- The default with JUnit is **PER\_METHOD** test class instantiation
- So when the above test class is executed, three separate instances of the class are created, one to run each of the three test methods
- This tends to make tests more independent
- A **PER\_CLASS** Lifecycle is also provided, if you need it

The default test instance  
lifecycle of one test class  
instance per test method is the  
safest and easiest mode

Other modes to keep state across multiple test  
method executions are possible, but typically not  
needed for true unit tests



# JUnit Lifecycle Methods

**Any test class method annotated by a JUnit lifecycle annotation**

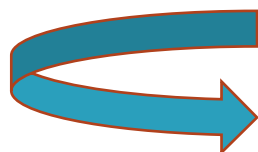
**These annotations are:**

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

**These annotations are used to create methods that can perform common setup or teardown code needed before and after tests run**



# How Lifecycle Methods Execute

  
Repeat per  
Test method

**@BeforeAll**

Executes once before all test methods in a given test class

**@BeforeEach**

Executes once before each test method

**@Test**

**@AfterEach**

Executes once after each test method

**@AfterAll**

Executes once after all test methods in a given test class



```
class MyTest {  
  
    @BeforeAll  
    static void initAll() {. . .}  
  
    @BeforeEach  
    void init() {. . .}  
  
    @Test  
    void test1() {. . .}  
  
    @Test  
    void test2() {. . .}  
  
    @AfterEach  
    void tearDown(. . .) { }  
  
    @AfterAll  
    static void tearDownAll() {. . .}  
  
}
```

◀ **initAll executes**

- ◀ - **init executes**
- ◀ - **test1 executes**
- ◀ - **tearDown executes**

- ◀ - **init executes**
- ◀ - **test2 executes**
- ◀ - **tearDown executes**

◀ **tearDownAll executes**



Mostly what you will use is a  
single @BeforeEach method



# Setting Up Tests with BeforeEach

---





# Creating Other Test Lifecycle Methods

---



# Up Next: Controlling Test Execution

---

