JUnit 5 Fundamentals

This course will teach you how to write unit tests with JUnit 5, covering topics like why unit tests are important, the structure of good unit tests, dynamic and parameterized tests, code coverage, and more.

Types of Testing

Acceptance

Integration

System

Load

Regression

Unit

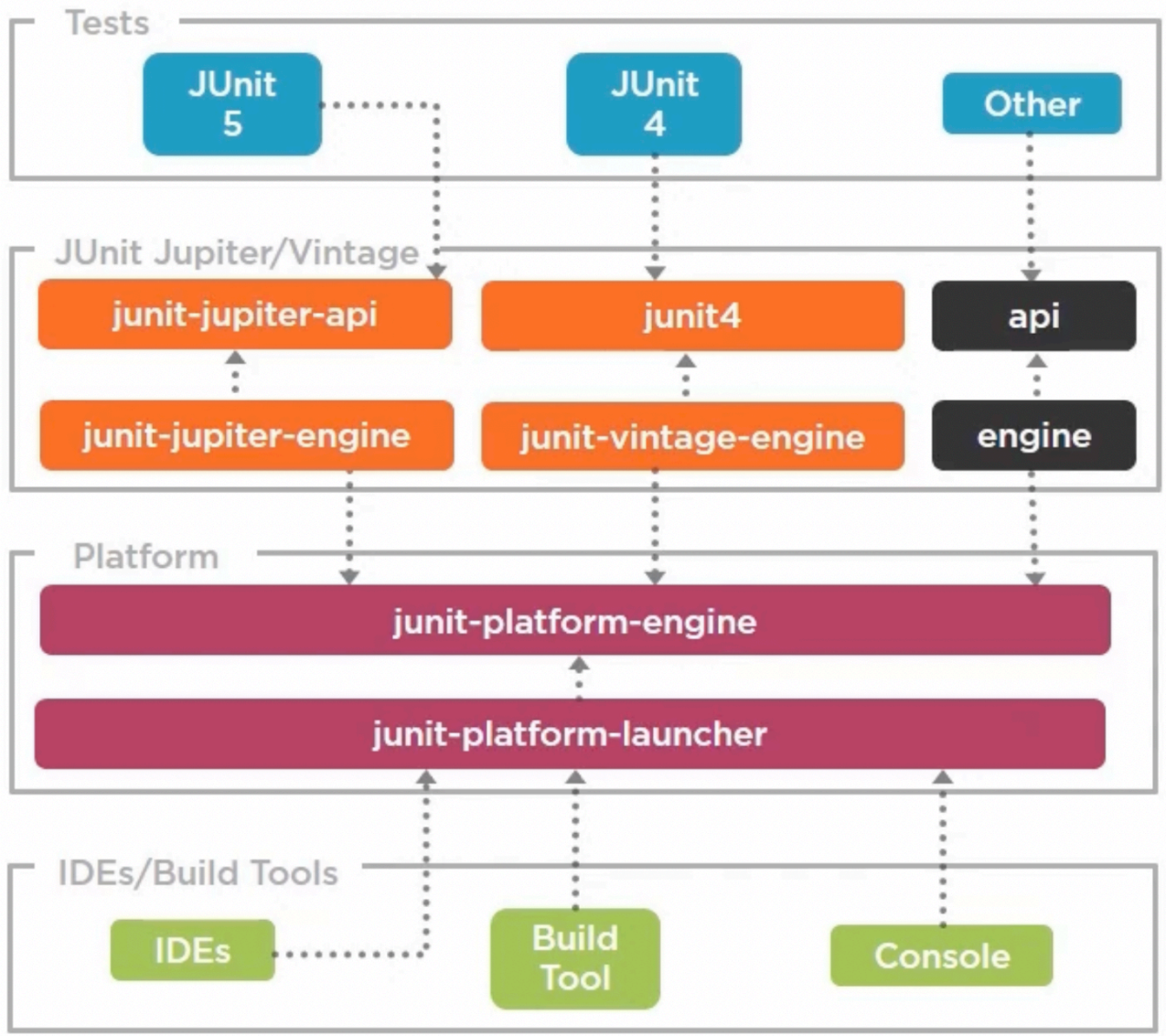
Stress

Security

Recovery

Usability

Architecture



What would we learn through this course?

- Architecture of JUnit 5
- How to use JUnit to create tests
- Integrate JUnit with Maven

Prerequisites:

- Basic knowledge of Java 8
- Streams, lambda expressions
- Default methods in interfaces

What are the most important and popular Java frameworks?

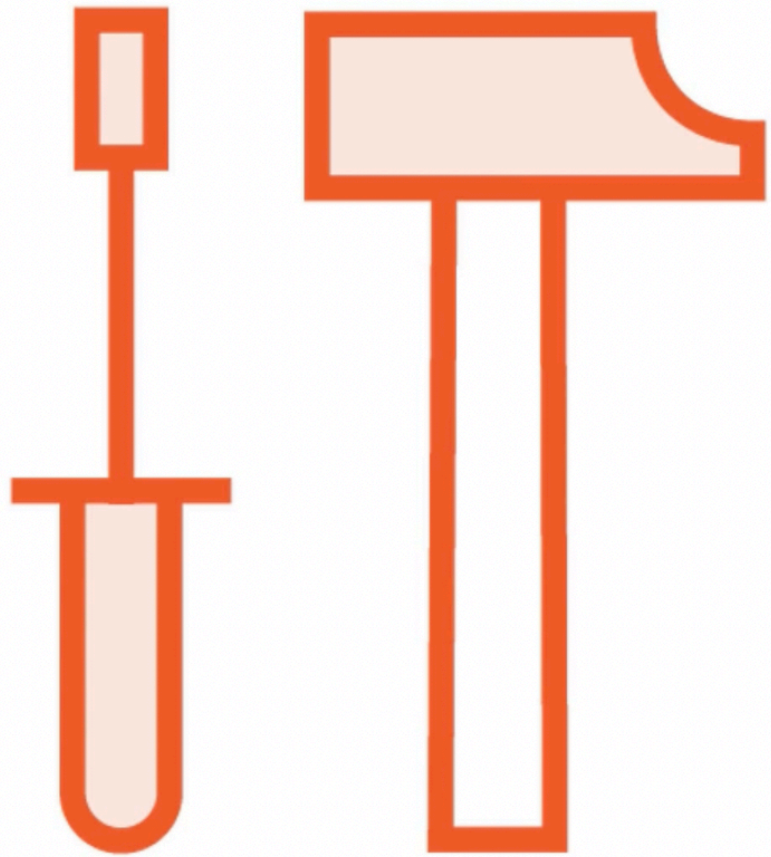
JUnit is the standard tool for unit testing in Java

“Never in the field of software development was so much owed by so many to so few lines of code.”

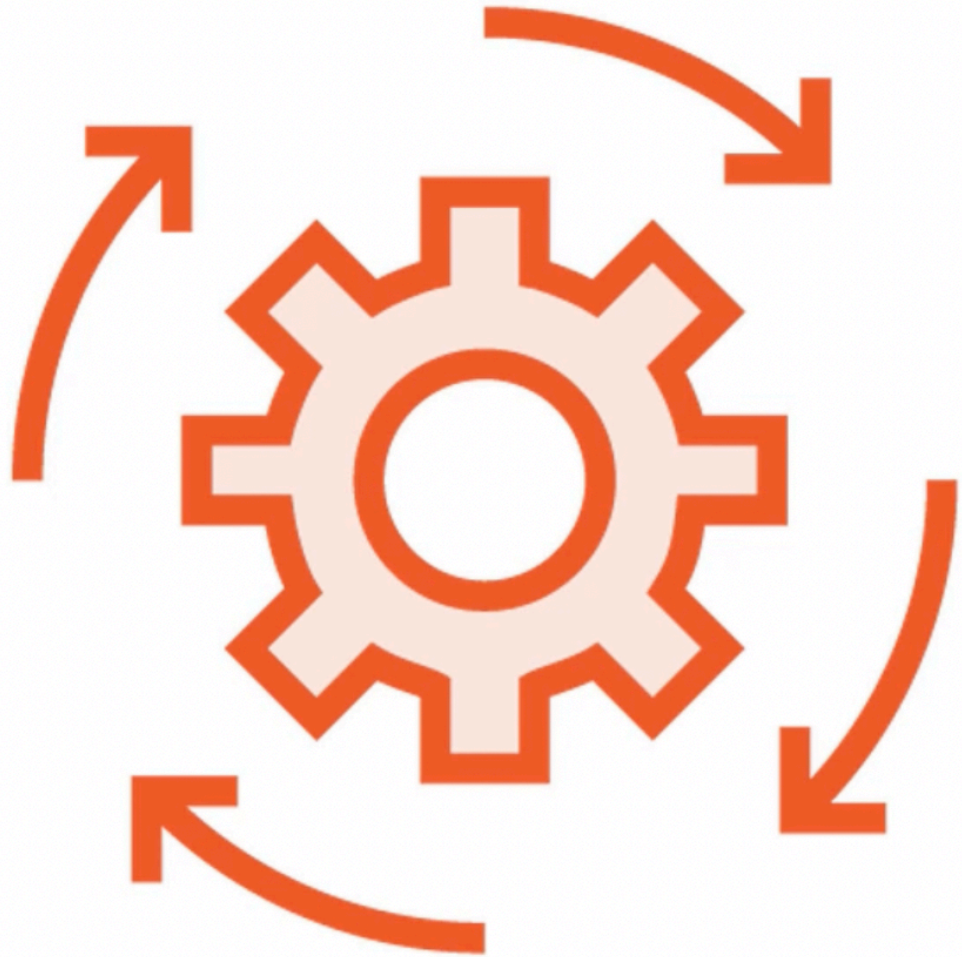
Martin Fowler

What is Junit?

JUnit Is...



Testing Is Feedback

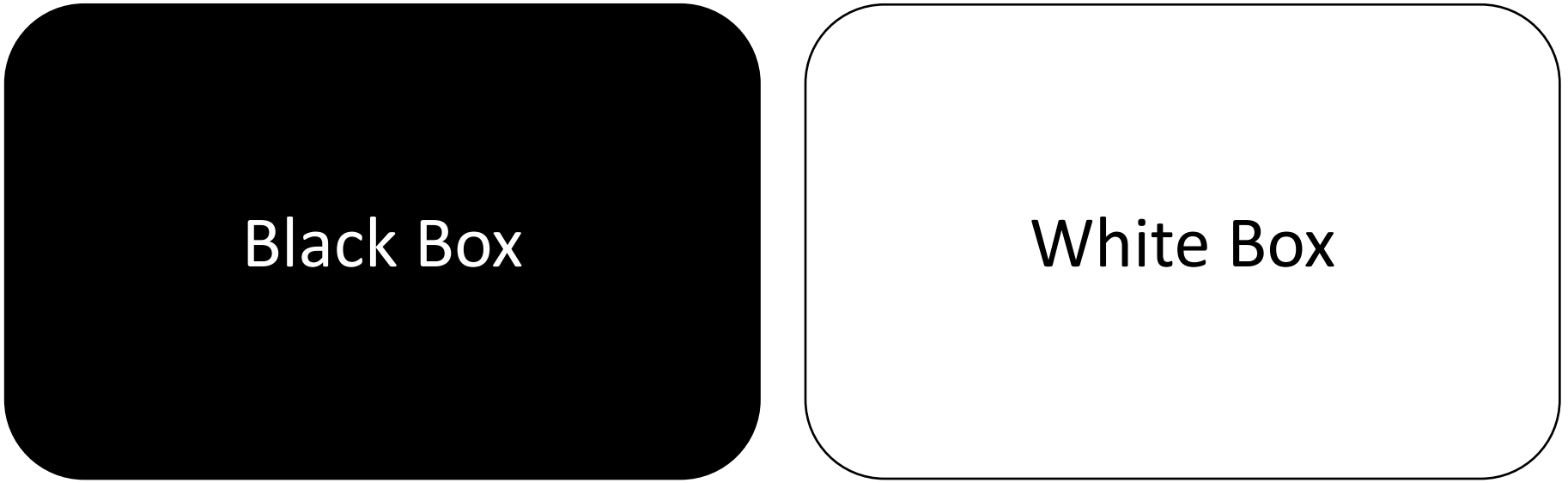


Tests Must Be…

- Easy to understand
- Easy to read
- Easy to modify

Types of Tests

According to the knowledge of the system



Types of Testing

Acceptance

Integration

System

Load

Regression

Unit

Stress

Security

Recovery

Usability

Types of Tests

Unit

Unit Tests

- Written by programmers
- Test a piece of code

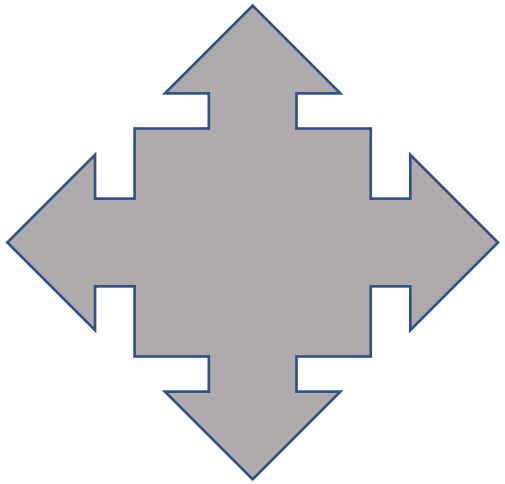
```
class Sale {
   ...
   public BigDecimal calculateCommission() {
      ...
   }
   ...
}
```

A Good Unit Test Should Be

- Automated
- Repeatable
- Fast

```
class Sale {
  private WebService commissionWS;
  ...
  public BigDecimal calculateCommission() {
    double commissionPercentage = commissionWS.getPercentage();
    ...
  }
  ...
}
```

Integration Tests



Test how components work together

```
void testCalculateCommission() {
    Sale sale = new Sale();
    sale.setCommission(new FakeWebService());

    // Test calculateCommission() method
}
```

Benefits

Reduce debugging time

Serve as documentation

Help to improve the design

Would It Be...

Hard?

Time consuming?

Well...

Yes, but not impossible

What If Our Unit Tests Were...

Easy to run

Fast

Complete

Up to date

Courage

What Do You Think Is Better?

Terrible designed system
Good suite of tests

Good designed system
Terrible suite of tests

First JUnit 5 Test

- Writing first JUnit 5 test
- Integrating JUnit 5 with maven

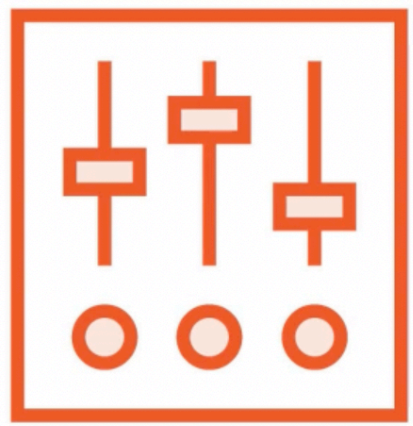
Course Scenario

Loyalty Program

- Every order generates points
- Rewards
  - Conversion
  - Discount
  - Gift

Four Phases of Every Test

Arrange



Act

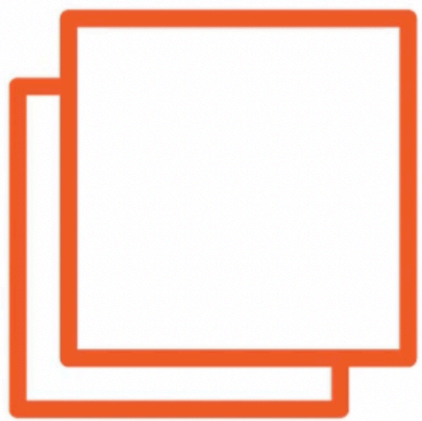
Assert

Annihilation

BDD Naming Style

Test Phases	BDD
Arrange	Given
Act	When
Assert	Then

Managing Test Fixtures



Transient Fresh

Persistence Fresh

Persistence Shared

Lifecycle Annotations

Once per method	Once per class
@BeforeEach	@BeforeAll
@AfterEach	@AfterAll

Lifecycle Execution

Per method (default)

Per class

JUnit Jupiter Assertions

- assertAll
- assertEquals
- assertFalse
- assertIterableEquals
- assertLinesMatch
- assertNotEquals
- assertNotNull
- assertNotSame
- assertNull
- assertSame
- assertThrows
- assertDoesNotThrow
- assertTimeout
- assertTimeoutPreemptively
- assertTrue
- fail

Annotation-based Conditions

Enable

```
@EnabledOnOs({ LINUX, MAC })

@EnabledOnJre(JAVA_8)

@EnabledForJreRange(max=JAVA_10)

@EnabledIfSystemProperty(named="version",
                         matches="1.*")

@EnabledIfEnvironmentVariable(
            named="ENV",matches="*server")

@EnabledIf("methodName")
```

Disable

```
@DisabledOnOs(WINDOWS)

@DisabledOnJre({ JAVA_9, JAVA_10 })

@DisabledForJreRange(min=JAVA_11,max=JAVA_14)

@DisabledIfSystemProperty(named="dev",
                          matches="true")

@DisabledIfEnvironmentVariable(
                    named="ENV",matches="QA")

@DisabledIf("com.example.MyClass#myMethod")
```

Assumptions

Based on conditions

Don’t result in test failure like assertions

Abort the test

Summary

- Test Structure
- Life Cycle Methods
- Test Hierarchies
- Assertions
- Disabling Tests
- Assumptions
- Test Interface and Default Methods
- Repeated Tests

- Parameterized Tests
- Argument Sources

- Integrating tests with Maven

Q & A