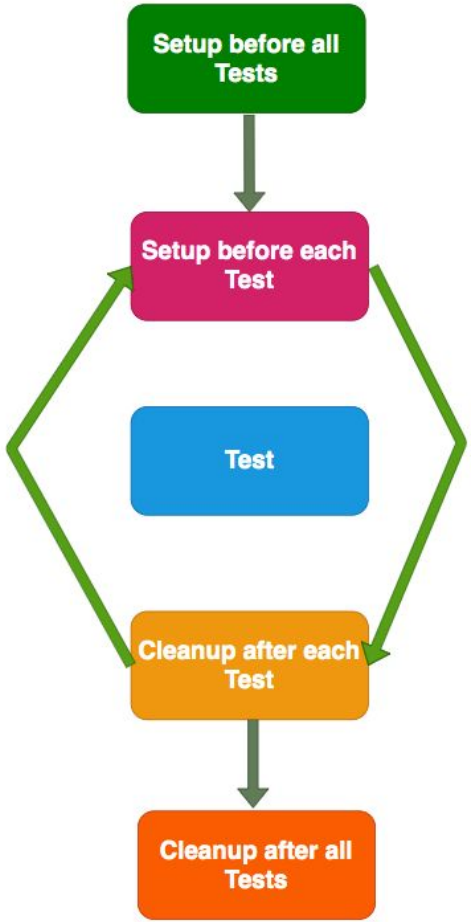
TDD with JUnit

Test-Driven Development (TDD)

- TDD Life Cycle
- TDD vs DLP (Debug Later Programming)
- Why TDD is matter
- Unit Testing with F.I.R.S.T
- Code and Test Coverage
- Structure of good unit testing (GUT)

Unit Testing with JUnit

- **JUnit lifeCycle**
- **Assertion**
- **Data-Driven Test with JUnit**
- **JUnit features**
- **Timeout**
- **Conditional**
- **Category**
- **Suite**
- **Running testing**



@BeforeAll

@BeforeEach

@Test

@AfterEach

@AfterAll

Assertions and assumptions

Assert statement	Example
assertEquals	assertEquals(4, calculator.multiply(2, 2),"optional failure message");
assertTrue	assertTrue('a' < 'b', () → "optional failure message");
assertFalse	assertFalse('a' > 'b', () → "optional failure message");
assertNotNull	assertNotNull(yourObject, "optional failure message");
assertNull	assertNull(yourObject, "optional failure message");

AssertTimeout

f you want to ensure that a test fails, if it isn't done in a certain amount of time you can use the `assertTimeout()` method. This assert fails the method if the timeout is exceeded.

COPY JAVA

```java
import static org.junit.jupiter.api.Assertions.assertTimeout;
import static java.time.Duration.ofSeconds;
import static java.time.Duration.ofMinutes;

@Test
void timeoutNotExceeded() {
    assertTimeout(ofMinutes(1), () -> service.doBackup());
}

// if you have to check a return value
@Test
void timeoutNotExceededWithResult() {
    String actualResult = assertTimeout(ofSeconds(1), () -> {
        return restService.request(request);
    });
    assertEquals(200, request.getStatus());
}
```

```
=> org.opentest4j.AssertionFailedError: execution exceeded timeout of
1000 ms by 212 ms
```

Testing for exceptions

Testing that certain exceptions are thrown are be done with the `org.junit.jupiter.api.Assertions.expectThrows()` assert statement. You define the expected Exception class and provide code that should throw the exception.

COPY JAVA

```java
import static org.junit.jupiter.api.Assertions.assertThrows;

@Test
void exceptionTesting() {
    // set up user
    Throwable exception = assertThrows(IllegalArgumentException.class,
() -> user.setAge("23"));
    assertEquals("Age must be an Integer.", exception.getMessage());
}
```

JUnit Features

Conditions

Operating System Conditions

```
@Test
@EnabledOnOs({OS.WINDOWS, OS.MAC})
public void shouldRunBothWindowsAndMac() {
    //...
}
```

```
@Test
@DisabledOnOs(OS.LINUX)
public void shouldNotRunAtLinux() {
    //...
}
```

JUnit Features

Conditions

Java Runtime Environment Conditions

```
@Test
@EnabledOnJre({JRE.JAVA_10, JRE.JAVA_11})
public void shouldOnlyRunOnJava10And11() {
    //...
}
```

```
@Test
@EnabledForJreRange(min = JRE.JAVA_8, max = JRE.JAVA_13)
public void shouldOnlyRunOnJava8UntilJava13() {
    // this test will only run on Java 8, 9, 10, 11, 12, and 13.
}
```

```
@Test
@DisabledForJreRange(min = JRE.JAVA_14, max = JRE.JAVA_15)
public void shouldNotBeRunOnJava14AndJava15() {
    // this won't run on Java 14 and 15.
}
```

JUnit Features

Conditions

System Property Conditions

```
@Test
@EnabledIfSystemProperty(named = "java.vm.vendor", matches = "Oracle.*")
public void onlyIfVendorNameStartsWithOracle() {
    //...
}
```

```
@Test
@DisabledIfSystemProperty(named = "file.separator", matches = "[/]")
public void disabledIfFileSeperatorIsSlash() {
    //...
}
```

JUnit Features

Conditions

Environment Variable Conditions

```
@Test
@EnabledIfEnvironmentVariable(named = "GDMSESSION", matches = "ubuntu")
public void onlyRunOnUbuntuServer() {
    //...
}

@Test
@DisabledIfEnvironmentVariable(named = "LC_TIME", matches = ".*UTF-8.")
public void shouldNotRunWhenTimeIsNotUTF8() {
    //...
}
```

What is not Unit?

External service

File system

Database

WebService API calls

UI specification

Compare screen image

Compare HTML

Test Name

```
public class FizzBuzzTest {

    @Test
    public void sayFizzWhenNumberIsDevidedByThree() {
        FizzBuzz fizzBuzz = new FizzBuzz();
        String actualResult = fizzBuzz.say(3);
        assertEquals("Fizz", actualResult);
    }

}
```

Annotation

Test Case Name

Test Naming

“What ’s in a name ?”

That which we call a rose

by any other name would smell

as sweet

Romeo and Juliet

Guide to Test Writing

Don’t say “**test**”, say “**should**”

Don’t use the word “test”

Use the word “should”

```
tranferShouldDeductSumFromSourceAccountBalance()

transferShouldAddSumLessFeesToDestinationAccountBalance()

depositShouldAddAmountToAccountBalance()
```

Guide to Test Writing

Don’t test your class, test **behaviour**

Guide to Test Writing

Test class names are important too

Structure your test well

Tests are **deliverable** too

Workshop#3

FizzBuzz

Input	Expected Result
1	1
2	2
3	Fizz
4	4
5	Buzz
6	Fizz
7	7
8	8
9	Fizz
10	Buzz
15	FizzBuzz

Good Unit Test

Arrange

Act

Assert

```
@Test
public void sayFizzWhenNumberIsDevidedByThree() {
    FizzBuzz fizzBuzz = new FizzBuzz();

    String actualResult = fizzBuzz.say(3);

    assertEquals("Fizz", actualResult);
}
```

Setup Pattern

```
@Test
public void sayFizzWhenNumberIsDevidedByThree() {
    FizzBuzz fizzBuzz = new FizzBuzz();

    String actualResult = fizzBuzz.say(3);

    assertEquals("Fizz", actualResult);
}

@Test
public void sayBuzzWhenNumberIsDevidedByFive() {
    FizzBuzz fizzBuzz = new FizzBuzz();

    String actualResult = fizzBuzz.say(5);

    assertEquals("Buzz", actualResult);
}
```

Setup Pattern

Inline Setup

```
@Test
public void sayFizzWhenNumberIsDevidedByThree() {
    FizzBuzz fizzBuzz = new FizzBuzz();

    String actualResult = fizzBuzz.say(3);

    assertEquals("Fizz", actualResult);
}

@Test
public void sayBuzzWhenNumberIsDevidedByFive() {
    FizzBuzz fizzBuzz = new FizzBuzz();

    String actualResult = fizzBuzz.say(5);

    assertEquals("Buzz", actualResult);
}
```

Setup Pattern

Delegate Setup

```
@Test
public void sayFizzWhenNumberIsDevidedByThree() {
    FizzBuzz fizzBuzz = setup();
    String actualResult = fizzBuzz.say(3);
    assertEquals("Fizz", actualResult);
}

@Test
public void sayBuzzWhenNumberIsDevidedByFive() {
    FizzBuzz fizzBuzz = setup();
    String actualResult = fizzBuzz.say(5);
    assertEquals("Buzz", actualResult);
}

private FizzBuzz setup() {
    FizzBuzz fizzBuzz = new FizzBuzz();
    return fizzBuzz;
}
```

Setup Pattern

```
@Before
public void initial() {
    fizzBuzz = new FizzBuzz();
}
```

Implicit Setup

```
@Test
public void sayFizzWhenNumberIsDevidedByThree() {
    String actualResult = fizzBuzz.say(3);
    assertEquals("Fizz", actualResult);
}

@Test
public void sayBuzzWhenNumberIsDevidedByFive() {
    String actualResult = fizzBuzz.say(5);
    assertEquals("Buzz", actualResult);
}
```

Implicit Teardown

```
@After
public void clearResources(){
    fizzBuzz = null;
}
```

Ignore Testcase

```
@Ignore("Pending implemetation")
@Test
public void sayFizzWhenNumberIsDevidedByThree() {
    FizzBuzz fizzBuzz = new FizzBuzz();
    String actualResult = fizzBuzz.say(3);
    assertEquals("Fizz", actualResult);
}
```

Handle Exception

Traditional approach

```
@Test
public void invalidWhenNumberLessThanOne() {
    try {
        fizzBuzz.say(0);
        fail();
    } catch (IllegalStateException expected) {
    }
}
```

Expected Annotation

```
@Test(expected=IllegalStateException.class)
public void invalidWhenNumberLessThanOne() {
        fizzBuzz.say(0);
}
```

ExpectedException Rule

```
@Rule
public ExpectedException thrown = ExpectedException.none();

@Test
public void invalidWhenNumberLessThanOne() {
    thrown.expect(IllegalStateException.class);
    fizzBuzz.say(0);
}
```

Data-Driven with JUnit

Using parameterized

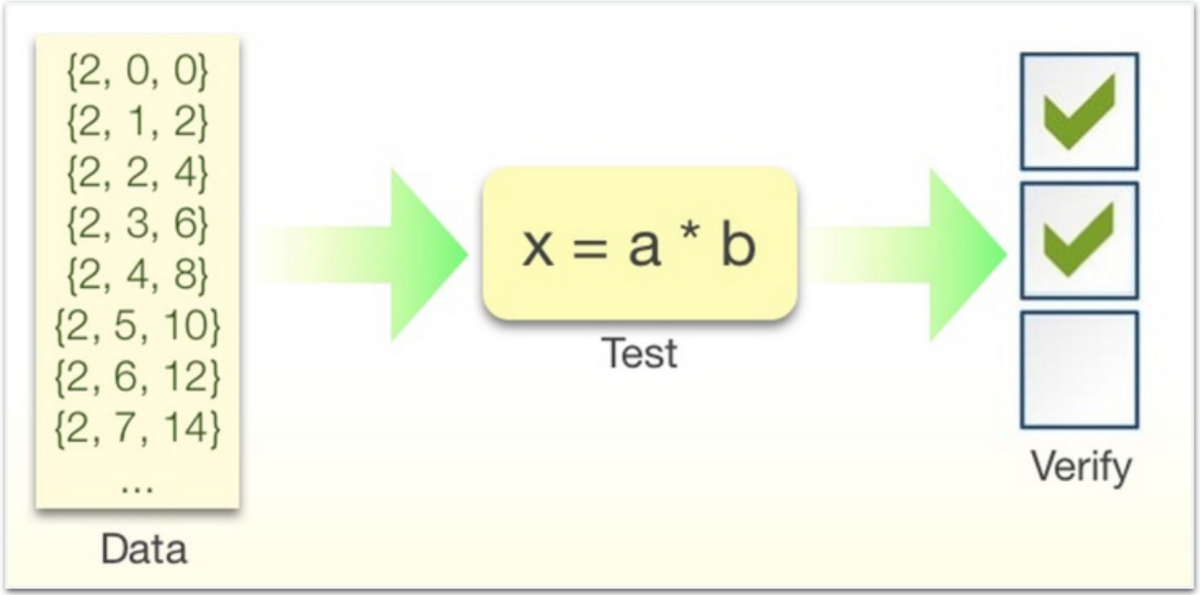
Set of test data

Expected result

Define test that uses the test data

Verify result against expected result

Data-Driven Development



@Parameterized
with Calculate Grade

Input	Expected Result
80	A
70	B
60	C
50	D
40	F

Step 1 :: Test Data

Input	Expected Result
80	A
70	B
60	C
50	D
40	F

Step 2 :: Add Runner

```
@RunWith(Parameterized.class)
public class CalculateGradeTest {

    private int score;
    private String expectedGrade;

    public CalculateGradeTest(int score, String expectedGrade) {
        this.score = score;
        this.expectedGrade = expectedGrade;
    }

    @Test
    public void convertScoreToGrade() {
    }

}
```

Step 3 :: Matching fields

```
@RunWith(Parameterized.class)
public class CalculateGradeTest {

    private int score;
    private String expectedGrade;

    public CalculateGradeTest(int score, String expectedGrade) {
        this.score = score;
        this.expectedGrade = expectedGrade;
    }

    @Test
    public void convertScoreToGrade() {
    }

}
```

Step 4 :: Define test case

```
@RunWith(Parameterized.class)
public class CalculateGradeTest {

    private int score;
    private String expectedGrade;

    public CalculateGradeTest(int score, String expectedGrade) {
        this.score = score;
        this.expectedGrade = expectedGrade;
    }

    @Test
    public void convertScoreToGrade() {
    }

}
```

Step 5 :: Add @parameters

```
@RunWith(Parameterized.class)
public class CalculateGradeTest {

    private int score;
    private String expectedGrade;

    @Parameters
    public static Collection<Object[]> data(){
        return Arrays.asList(new Object[][]{
                {80, "A"},
                {70, "B"},
                {60, "C"},
                {50, "D"},
                {40, "F"}
        });
    }
```

Workshop

@Parameterized
with Calculator

Operand 1	Operator	Operand 2	Expected
1	+	1	2
2	+	2	4
1	-	1	0
2	-	1	1
3	*	1	3
3	*	2	6