BridgeLabz

Employability Delivered

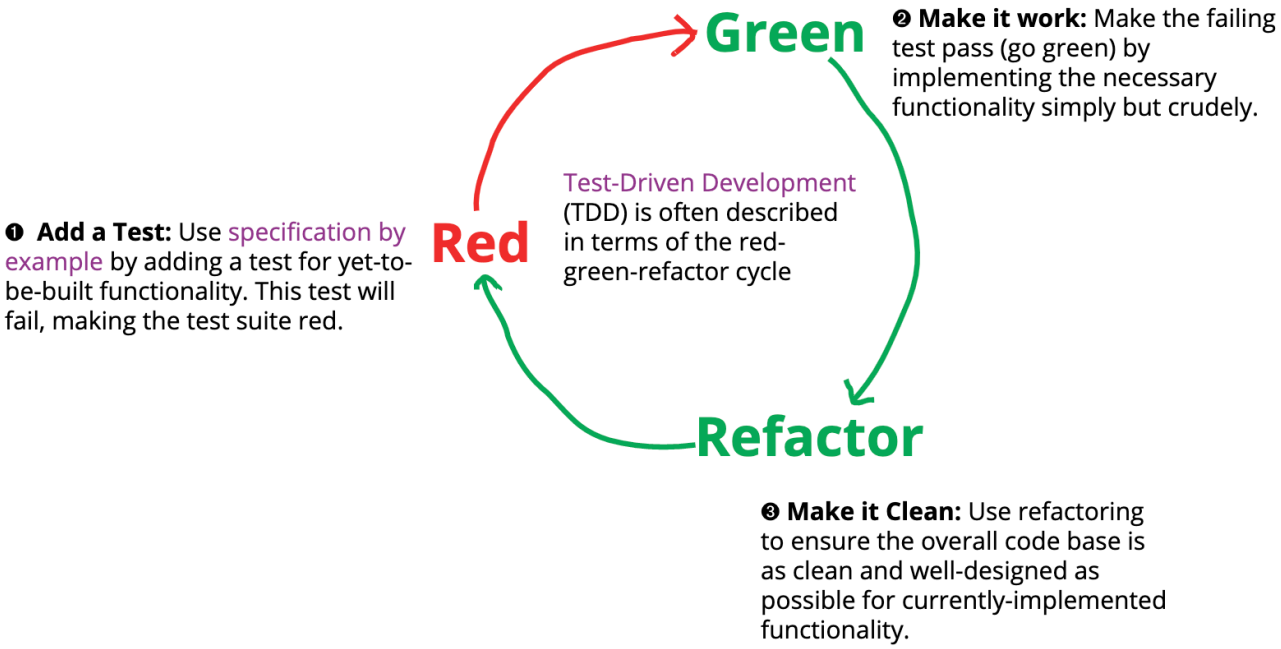
Junit Intro

TDD Intro – Red Green Refactor

BridgeLabz
Employability Delivered

martinFowler.com 2/13

Refactoring is often taught in the context of TDD



Junit – Tips and Intro

BridgeLabz
Employability Delivered

Unit testing tips:

- The entire goal is **FAILURE ATOMICITY**- the ability to know exactly what failed when a test case did not pass
- Tests should be self-contained and not care about each other
- you cannot test everything! Instead think about:
  - boundary cases,
  - empty cases,
  - behavior in combination (but not to excess)
- Each test case should test ONE THING
  - 10 small tests are better than 1 test 10x as large
  - Rule of thumb: 1 assert statement per test case
  - Try to avoud complicated logic
- Torture tests are ok, but only *in addition* to simple tests

Junit – Tips and Intro

BridgeLabz
Employability Delivered

Unit testing tips:

- The entire goal is **FAILURE ATOMICITY**- the ability to know exactly what failed when a test case did not pass
- Tests should be self-contained and not care about each other
- you cannot test everything! Instead think about:
  - boundary cases,
  - empty cases,
  - behavior in combination (but not to excess)
- Each test case should test ONE THING
  - 10 small tests are better than 1 test 10x as large
  - Rule of thumb: 1 assert statement per test case
  - Try to avoud complicated logic
- Torture tests are ok, but only *in addition* to simple tests

Junit – Tips and Intro

BridgeLabz
Employability Delivered

JUnit 4

Method annotations:

tag	description

`@Test`
`@Test (timeout = time)`
`@Test (expected = exception.class)`
	Turns a public method into a JUnit test case. Adding a timeout will cause the test case to fail after **time** milliseconds. Adding an expected exception will cause the test case to fail if **exception** is not thrown.

`@Before`	Method to run before every test case
`@After`	Method to run after every test case
`@BeforeClass`	Method to run once, before any test cases have run
`@AfterClass`	Method to run once, after all test cases have run

Assertion methods:

method	description
`assertTrue(test)`	fails if the Boolean test is `false`
`assertFalse(test)`	fails if the Boolean test is `true`
`assertEquals(expected, actual)`	fails if the values are not equal
`assertSame(expected, actual)`	fails if the values are not the same (by ==)
`assertNotSame(expected, actual)`	fails if the values are the same (by ==)
`assertNull(value)`	fails if the given value is not `null`
`assertNotNull(value)`	fails if the given value is `null`
`fail()`	causes the current test to immediately fail

Test for Boolean

Test for Equality

Test for Identical Objects

Test for Nullability Objects

Each method can also be passed a string to display if it fails, e.g.
`assertEquals("message", expected, actual)`

Junit – Tips and Intro

```java
package com.dummyproject;

import org.junit.Assert;
import org.junit.Test;

public class Junit4AssertionTest {
    @Test
    public void testAssert(){

        //Variable declaration
        String string1="Junit";
        String string2="Junit";
        Object obj1 = new Object();
        Object obj2 = new Object();
        String string5=null;
        int variable1=1;
        int variable2=2;
        int[] airethematicArrary1 = { 1, 2, 3 };
        int[] airethematicArrary2 = { 1, 2, 3 };

        //Assert statements
        Assert.assertEquals(string1,string2);
        Assert.assertSame(string1,string2);
        Assert.assertSame(obj1, obj1);
        Assert.assertNotSame(obj1, obj2);
        Assert.assertNotNull(string1);
        Assert.assertNull(string5);
        Assert.assertTrue( condition: variable1<variable2);
        Assert.assertArrayEquals(airethematicArrary1, airethematicArrary2);
    }
}
```

BridgeLabz
Employability Delivered

Junit Parameterized Test

BridgeLabz
Employability Delivered

Parameterized test is to execute the same test over and over again using different values. It helps developer to save time in executing same test which differs only in their inputs and expected results.

Junit – Parametrized Test

```java
package junitTutorial;

import static org.junit.Assert.assertEquals;
import java.util.Arrays;
import java.util.Collection;
import org.junit.Before;
import org.junit.Test;
import org.junit.runner.RunWith;
import org.junit.runners.Parameterized;

@RunWith(Parameterized.class)
public class AirthematicTest {
    private int firstNumber;
    private int secondNumber;
    private int expectedResult;
    private Airthematic airthematic;

    public AirthematicTest(int firstNumber, int secondNumber, int expectedResult) {
        super();
        this.firstNumber = firstNumber;
        this.secondNumber = secondNumber;
        this.expectedResult = expectedResult;
    }

    @Before
    public void initialize() {
        airthematic = new Airthematic();
    }

    @Parameterized.Parameters
    public static Collection input() {
        return Arrays.asList(new Object[][] { { 1, 2, 3 }, { 11, 22, 33 },
                { 111, 222, 333 }, { 10, 9, 19 }, { 100, 9, 109 } });
    }

    @Test
    public void testAirthematicTest() {
        System.out.println("Sum of Numbers = : " + expectedResult);
        assertEquals(expectedResult, airthematic.sum(firstNumber, secondNumber));
    }
```

BridgeLabz

Employability Delivered

Thankyou