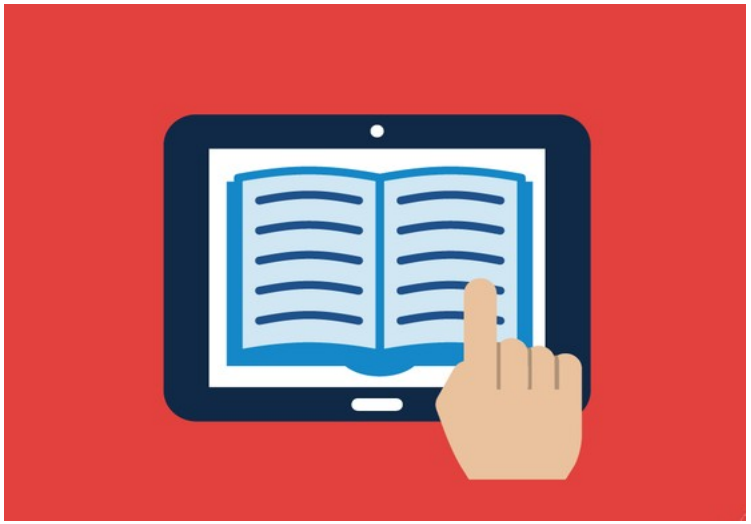
Clean Unit Test Patterns

Frank Appel

Blog: www.codeaffine.com
Email: fappel@codeaffine.com
Twitter: @frank_appel

Clean Unit Test Patterns



Why bother?

Structure

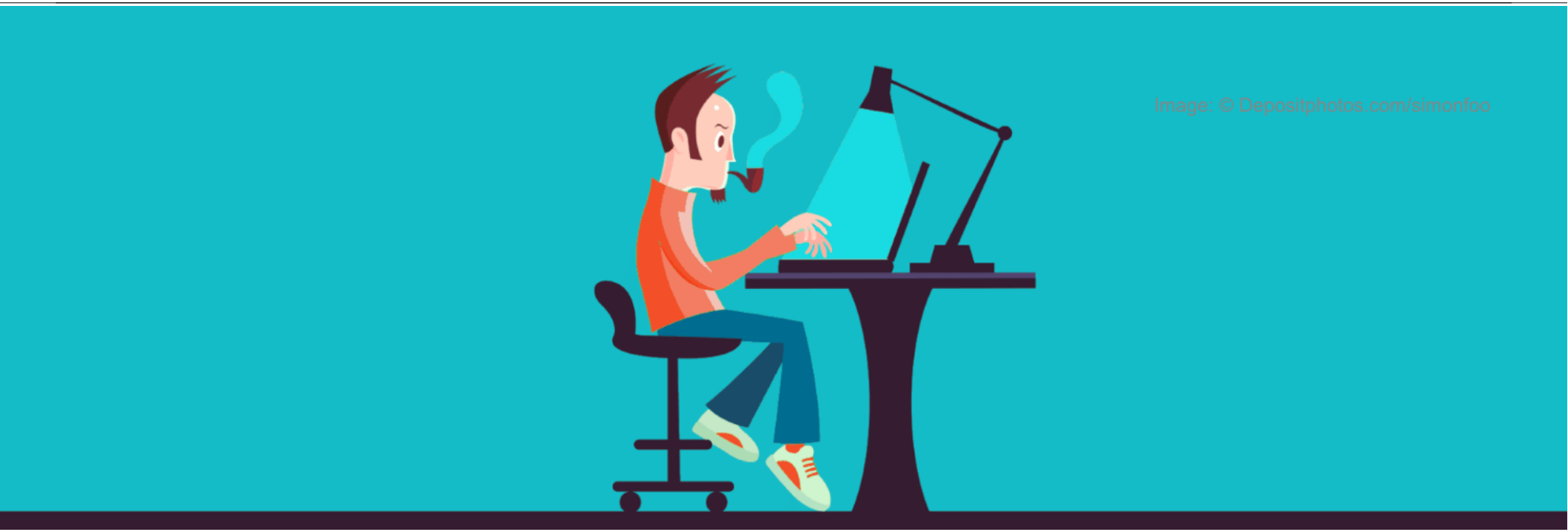
Isolation

Runners and Rules

Assertions

Q&A

Who I am..

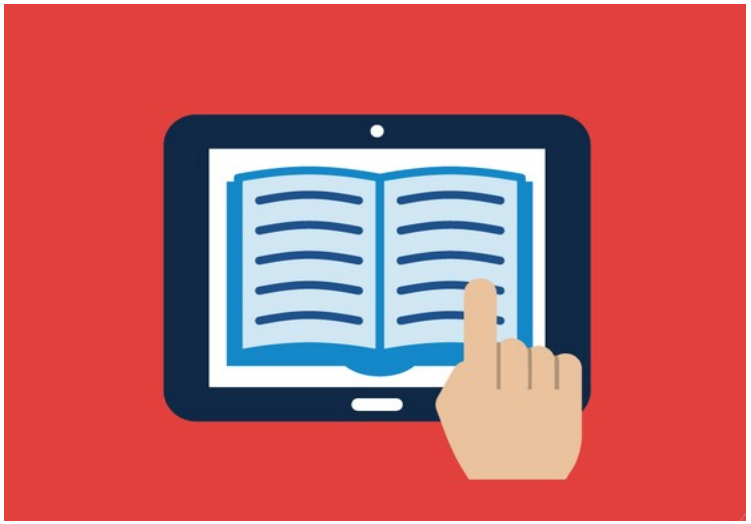


Independent Software Developer

Blogger (http://codeaffine.com/blog)

Stalwart of agile methods and TDD in particular

Clean Unit Test Patterns



Why bother?

Structure

Isolation

Runners and Rules

Assertions

Q&A

Why bother?

```
@Test
 public void testGetImage() {
   Fixture.useDefaultResourceManager();
   ResourceManager resourceManager = RWT.getResourceManager();
   // only if you comment initial registration in
   // org.eclipse.swt.internal.widgets.displaykit.QooxdooResourcesUtil
   assertFalse( resourceManager.isRegistered( Fixture.IMAGE1 ) );
   Image image1 = Graphics.getImage( Fixture.IMAGE1 );
   String registerPath = getRegisterPath( image1 );
   assertTrue( resourceManager.isRegistered( registerPath ) );
   File contextDir = new File( Fixture.WEB_CONTEXT_DIR, ResourceDirectory.DIRNAME );
   assertTrue( new File( contextDir, registerPath ).exists() );
   Image image1a = Graphics.getImage( Fixture.IMAGE1 );
   assertSame( image1, image1a );
   // another picture
   Image image2 = Graphics.getImage( Fixture.IMAGE2 );
   String image2Path = getRegisterPath( image2 );
   assertTrue( resourceManager.isRegistered( image2Path ) );
   assertTrue( new File( contextDir, image2Path ).exists() );
   // ... and do it again...
   Graphics.getImage( Fixture.IMAGE1 );
   assertTrue( resourceManager.isRegistered( registerPath ) );
 }
```

Why bother?

Test of too many concepts

```
@Test
 public void testGetImage() {
   Fixture.useDefaultResourceManager();
   ResourceManager resourceManager = RWT.getResourceManager();
   // only if you comment initial registration in
   // org.eclipse.swt.internal.widgets.displaykit.QooxdooResourcesUtil
   assertFalse( resourceManager.isRegistered( Fixture.IMAGE1 ) );
   Image image1 = Graphics.getImage( Fixture.IMAGE1 );
   String registerPath = getRegisterPath( image1 );
   assertTrue( resourceManager.isRegistered( registerPath ) );
   File contextDir = new File( Fixture.WEB_CONTEXT_DIR, ResourceDirectory.DIRNAME );
   assertTrue( new File( contextDir, registerPath ).exists() );
   Image image1a = Graphics.getImage( Fixture.IMAGE1 );
   assertSame( image1, image1a );
   // another picture
   Image image2 = Graphics.getImage( Fixture.IMAGE2 );
   String image2Path = getRegisterPath( image2 );
   assertTrue( resourceManager.isRegistered( image2Path ) );
   assertTrue( new File( contextDir, image2Path ).exists() );
   // ... and do it again...
   Graphics.getImage( Fixture.IMAGE1 );
   assertTrue( resourceManager.isRegistered( registerPath ) );
 }
```

Why bother?

```
@Test
 public void testGetImage() {
   Fixture.useDefaultResourceManager();
   ResourceManager resourceManager = RWT.getResourceManager();
   // only if you comment initial registration in
   // org.eclipse.swt.internal.widgets.displaykit.QooxdooResourcesUtil
   assertFalse( resourceManager.isRegistered( Fixture.IMAGE1 ) );
   Image image1 = Graphics.getImage( Fixture.IMAGE1 );
   String registerPath = getRegisterPath( image1 );
   assertTrue( resourceManager.isRegistered( registerPath ) );
   File contextDir = new File( Fixture.WEB_CONTEXT_DIR, ResourceDirectory.DIRNAME );
   assertTrue( new File( contextDir, registerPath ).exists() );
   Image image1a = Graphics.getImage( Fixture.IMAGE1 );
   assertSame( image1, image1a );
   // another picture
   Image image2 = Graphics.getImage( Fixture.IMAGE2 );
   String image2Path = getRegisterPath( image2 );
   assertTrue( resourceManager.isRegistered( image2Path ) );
   assertTrue( new File( contextDir, image2Path ).exists() );
   // ... and do it again...
   Graphics.getImage( Fixture.IMAGE1 );
   assertTrue( resourceManager.isRegistered( registerPath ) );
 }
```

Why bother?

Mix of integration and unit test

```
@Test
 public void testGetImage() {
   Fixture.useDefaultResourceManager();
   ResourceManager resourceManager = RWT.getResourceManager();
   // only if you comment initial registration in
   // org.eclipse.swt.internal.widgets.displaykit.QooxdooResourcesUtil
   assertFalse( resourceManager.isRegistered( Fixture.IMAGE1 ) );
   Image image1 = Graphics.getImage( Fixture.IMAGE1 );
   String registerPath = getRegisterPath( image1 );
   assertTrue( resourceManager.isRegistered( registerPath ) );
   File contextDir = new File( Fixture.WEB_CONTEXT_DIR, ResourceDirectory.DIRNAME );
   assertTrue( new File( contextDir, registerPath ).exists() );
   Image image1a = Graphics.getImage( Fixture.IMAGE1 );
   assertSame( image1, image1a );
   // another picture
   Image image2 = Graphics.getImage( Fixture.IMAGE2 );
   String image2Path = getRegisterPath( image2 );
   assertTrue( resourceManager.isRegistered( image2Path ) );
   assertTrue( new File( contextDir, image2Path ).exists() );
   // ... and do it again...
   Graphics.getImage( Fixture.IMAGE1 );
   assertTrue( resourceManager.isRegistered( registerPath ) );
 }
```

Why bother?

```
@Test
 public void testGetImage() {
   Fixture.useDefaultResourceManager();
   ResourceManager resourceManager = RWT.getResourceManager();
   // only if you comment initial registration in
   // org.eclipse.swt.internal.widgets.displaykit.QooxdooResourcesUtil
   assertFalse( resourceManager.isRegistered( Fixture.IMAGE1 ) );
   Image image1 = Graphics.getImage( Fixture.IMAGE1 );
   String registerPath = getRegisterPath( image1 );
   assertTrue( resourceManager.isRegistered( registerPath ) );
   File contextDir = new File( Fixture.WEB_CONTEXT_DIR, ResourceDirectory.DIRNAME );
   assertTrue( new File( contextDir, registerPath ).exists() );
   Image image1a = Graphics.getImage( Fixture.IMAGE1 );
   assertSame( image1, image1a );
   // another picture
   Image image2 = Graphics.getImage( Fixture.IMAGE2 );
   String image2Path = getRegisterPath( image2 );
   assertTrue( resourceManager.isRegistered( image2Path ) );
   assertTrue( new File( contextDir, image2Path ).exists() );
   // ... and do it again...
   Graphics.getImage( Fixture.IMAGE1 );
   assertTrue( resourceManager.isRegistered( registerPath ) );
 }
```

Why bother?

Missing of clean and recognizable test structure

```
@Test
 public void testGetImage() {
   Fixture.useDefaultResourceManager();
   ResourceManager resourceManager = RWT.getResourceManager();
   // only if you comment initial registration in
   // org.eclipse.swt.internal.widgets.displaykit.QooxdooResourcesUtil
   assertFalse( resourceManager.isRegistered( Fixture.IMAGE1 ) );
   Image image1 = Graphics.getImage( Fixture.IMAGE1 );
   String registerPath = getRegisterPath( image1 );
   assertTrue( resourceManager.isRegistered( registerPath ) );
   File contextDir = new File( Fixture.WEB_CONTEXT_DIR, ResourceDirectory.DIRNAME );
   assertTrue( new File( contextDir, registerPath ).exists() );
   Image image1a = Graphics.getImage( Fixture.IMAGE1 );
   assertSame( image1, image1a );
   // another picture
   Image image2 = Graphics.getImage( Fixture.IMAGE2 );
   String image2Path = getRegisterPath( image2 );
   assertTrue( resourceManager.isRegistered( image2Path ) );
   assertTrue( new File( contextDir, image2Path ).exists() );
   // ... and do it again...
   Graphics.getImage( Fixture.IMAGE1 );
   assertTrue( resourceManager.isRegistered( registerPath ) );
 }
```

Why bother?

```
@Test
 public void testGetImage() {
   Fixture.useDefaultResourceManager();
   ResourceManager resourceManager = RWT.getResourceManager();
   // only if you comment initial registration in
   // org.eclipse.swt.internal.widgets.displaykit.QooxdooResourcesUtil
   assertFalse( resourceManager.isRegistered( Fixture.IMAGE1 ) );
   Image image1 = Graphics.getImage( Fixture.IMAGE1 );
   String registerPath = getRegisterPath( image1 );
   assertTrue( resourceManager.isRegistered( registerPath ) );
   File contextDir = new File( Fixture.WEB_CONTEXT_DIR, ResourceDirectory.DIRNAME );
   assertTrue( new File( contextDir, registerPath ).exists() );
   Image image1a = Graphics.getImage( Fixture.IMAGE1 );
   assertSame( image1, image1a );
   // another picture
   Image image2 = Graphics.getImage( Fixture.IMAGE2 );
   String image2Path = getRegisterPath( image2 );
   assertTrue( resourceManager.isRegistered( image2Path ) );
   assertTrue( new File( contextDir, image2Path ).exists() );
   // ... and do it again...
   Graphics.getImage( Fixture.IMAGE1 );
   assertTrue( resourceManager.isRegistered( registerPath ) );
 }
```

Why bother?

Tight coupling of unit under test and dependencies

```
@Test
 public void testGetImage() {
   Fixture.useDefaultResourceManager();
   ResourceManager resourceManager = RWT.getResourceManager();
   // only if you comment initial registration in
   // org.eclipse.swt.internal.widgets.displaykit.QooxdooResourcesUtil
   assertFalse( resourceManager.isRegistered( Fixture.IMAGE1 ) );
   Image image1 = Graphics.getImage( Fixture.IMAGE1 );
   String registerPath = getRegisterPath( image1 );
   assertTrue( resourceManager.isRegistered( registerPath ) );
   File contextDir = new File( Fixture.WEB_CONTEXT_DIR, ResourceDirectory.DIRNAME );
   assertTrue( new File( contextDir, registerPath ).exists() );
   Image image1a = Graphics.getImage( Fixture.IMAGE1 );
   assertSame( image1, image1a );
   // another picture
   Image image2 = Graphics.getImage( Fixture.IMAGE2 );
   String image2Path = getRegisterPath( image2 );
   assertTrue( resourceManager.isRegistered( image2Path ) );
   assertTrue( new File( contextDir, image2Path ).exists() );
   // ... and do it again...
   Graphics.getImage( Fixture.IMAGE1 );
   assertTrue( resourceManager.isRegistered( registerPath ) );
 }
```

Why bother?

```
@Test
 public void testGetImage() {
   Fixture.useDefaultResourceManager();
   ResourceManager resourceManager = RWT.getResourceManager();
   // only if you comment initial registration in
   // org.eclipse.swt.internal.widgets.displaykit.QooxdooResourcesUtil
   assertFalse( resourceManager.isRegistered( Fixture.IMAGE1 ) );
   Image image1 = Graphics.getImage( Fixture.IMAGE1 );
   String registerPath = getRegisterPath( image1 );
   assertTrue( resourceManager.isRegistered( registerPath ) );
   File contextDir = new File( Fixture.WEB_CONTEXT_DIR, ResourceDirectory.DIRNAME );
   assertTrue( new File( contextDir, registerPath ).exists() );
   Image image1a = Graphics.getImage( Fixture.IMAGE1 );
   assertSame( image1, image1a );
   // another picture
   Image image2 = Graphics.getImage( Fixture.IMAGE2 );
   String image2Path = getRegisterPath( image2 );
   assertTrue( resourceManager.isRegistered( image2Path ) );
   assertTrue( new File( contextDir, image2Path ).exists() );
   // ... and do it again...
   Graphics.getImage( Fixture.IMAGE1 );
   assertTrue( resourceManager.isRegistered( registerPath ) );
 }
```

Why bother?

Poor maintainability and progression

```
@Test
 public void testGetImage() {
   Fixture.useDefaultResourceManager();
   ResourceManager resourceManager = RWT.getResourceManager();
   // only if you comment initial registration in
   // org.eclipse.swt.internal.widgets.displaykit.QooxdooResourcesUtil
   assertFalse( resourceManager.isRegistered( Fixture.IMAGE1 ) );
   Image image1 = Graphics.getImage( Fixture.IMAGE1 );
   String registerPath = getRegisterPath( image1 );
   assertTrue( resourceManager.isRegistered( registerPath ) );
   File contextDir = new File( Fixture.WEB_CONTEXT_DIR, ResourceDirectory.DIRNAME );
   assertTrue( new File( contextDir, registerPath ).exists() );
   Image image1a = Graphics.getImage( Fixture.IMAGE1 );
   assertSame( image1, image1a );
   // another picture
   Image image2 = Graphics.getImage( Fixture.IMAGE2 );
   String image2Path = getRegisterPath( image2 );
   assertTrue( resourceManager.isRegistered( image2Path ) );
   assertTrue( new File( contextDir, image2Path ).exists() );
   // ... and do it again...
   Graphics.getImage( Fixture.IMAGE1 );
   assertTrue( resourceManager.isRegistered( registerPath ) );
 }
```

Why bother?

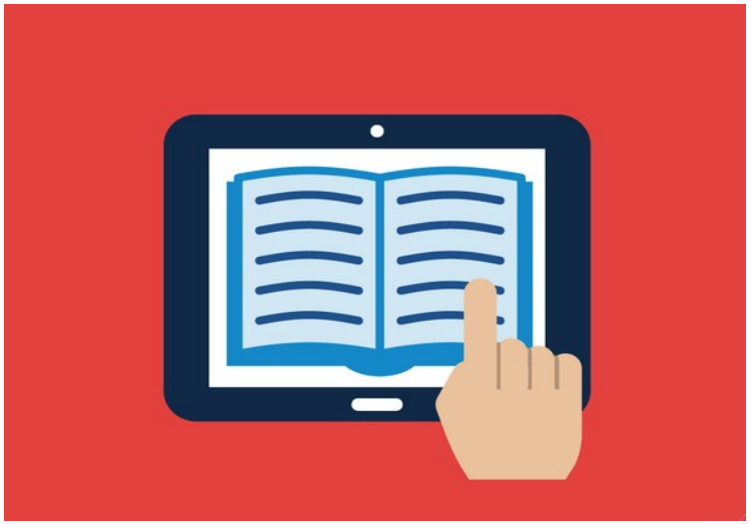
```
@Test
 public void testGetImage() {
   Fixture.useDefaultResourceManager();
   ResourceManager resourceManager = RWT.getResourceManager();
   // only if you comment initial registration in
   // org.eclipse.swt.internal.widgets.displaykit.QooxdooResourcesUtil
   assertFalse( resourceManager.isRegistered( Fixture.IMAGE1 ) );
   Image image1 = Graphics.getImage( Fixture.IMAGE1 );
   String registerPath = getRegisterPath( image1 );
   assertTrue( resourceManager.isRegistered( registerPath ) );
   File contextDir = new File( Fixture.WEB_CONTEXT_DIR, ResourceDirectory.DIRNAME );
   assertTrue( new File( contextDir, registerPath ).exists() );
   Image image1a = Graphics.getImage( Fixture.IMAGE1 );
   assertSame( image1, image1a );
   // another picture
   Image image2 = Graphics.getImage( Fixture.IMAGE2 );
   String image2Path = getRegisterPath( image2 );
   assertTrue( resourceManager.isRegistered( image2Path ) );
   assertTrue( new File( contextDir, image2Path ).exists() );
   // ... and do it again...
   Graphics.getImage( Fixture.IMAGE1 );
   assertTrue( resourceManager.isRegistered( registerPath ) );
 }
```

Why bother?

```
@Test
 public void testGetImag
   Fixture.useDefault  rceManager();
   ResourceManager  ourceManager = RWT.get  rceManager();
   // only if yo  mment initial registrat
   // org.eclip  swt.internal.widgets.di  ki  oxdooResourcesUtil
   assertFalse  esourceManager.isRegi  d( Fix  e.IMAGE1 ) );
   Image image  = Graphics.getImage  ure.IMAGE  );
   Strin  eg  erPath = getRegis  h( image1 )
   assertT ue  esourceManager  gistered( regist  Path ) );
   il  co  ir = new Fil  xture.WEB_CONTEXT  R,  ceDi  y.DIRNAME );
   assertTrue  w File(  tDir, registerPath  xists() );
   Image image  = Gra  getImage( Fixture.IMA  );
   assertSame(  ge1a );
   // another pic
   Image image2 = G  ics.getImage( Fixtur  GE2 );
   String image2Path  gisterPath(  2 );
   assertTrue( resourceMa  ed( image2Path ) );
   assertTrue( new File( contextDir, image2Path ).exists() );
   // ... and do it again...
   Graphics.getImage( Fixture.IMAGE1 );
   assertTrue( resourceManager.isRegistered( registerPath ) );
 }
```

Don't do it!

Clean Unit Test Patterns



Why bother?

Structure

Isolation

Runners and Rules

Assertions

Q&A

Test Structure

Four Phases Pattern

```
1  @Test
2  public void subsequentNumber() {
3    NumberRangeCounter counter = new NumberRangeCounter();
4
5    int first = counter.next();
6    int second = counter.next();
7
8    assertEquals( first + 1, second );
9  }
```

Test Structure

Four Phases Pattern

Setup (Fixture)

```
1  @Test
2  public void subsequentNumber() {
3    NumberRangeCounter counter = new NumberRangeCounter();
4
5    int first = counter.next();
6    int second = counter.next();
7
8    assertEquals( first + 1, second );
9  }
```

Test Structure

Four Phases Pattern

Setup (Fixture)

Exercise

```
1  @Test
2  public void subsequentNumber() {
3    NumberRangeCounter counter = new NumberRangeCounter();
4
5    int first = counter.next();
6    int second = counter.next();
7
8    assertEquals( first + 1, second );
9  }
```

Test Structure

Four Phases Pattern

Setup (Fixture)

Exercise

Verify

```
1  @Test
2  public void subsequentNumber() {
3    NumberRangeCounter counter = new NumberRangeCounter();
4
5    int first = counter.next();
6    int second = counter.next();
7
8    assertEquals( first + 1, second );
9  }
```

Test Structure

Four Phases Pattern

Setup (Fixture)

Exercise

Verify

```
1  @Test
2  public void subsequentNumber() {
3    NumberRangeCounter counter = new NumberRangeCounter();
4
5    int first = counter.next();
6    int second = counter.next();
7
8    assertEquals( first + 1, second );
9  }
```

Teardown: cleaning up the fixture in case it is persistent.

Test Structure

Four Phases Pattern

Setup (Fixture)

Exercise

Verify

```
1  @Test
2  public void subsequentNumber() {
3    NumberRangeCounter counter = new NumberRangeCounter();
4
5    int first = counter.next();
6    int second = counter.next();
7
8    assertEquals( first + 1, second );
9  }
```

Teardown: cleaning up the fixture in case it is persistent.

> *The ratio of time spent reading (code) versus writing is well over 10 to 1...*
>
> *Robert C. Martin, Clean Code*

Test Structure

Setup Patterns

```
01 public class NumberRangeCounterTest {
02
03   private static final int LOWER_BOUND = 1000;
04
05   @Test
06   public void subsequentNumber() {
07     NumberRangeCounter counter = new NumberRangeCounter( LOWER_BOUND );
08
09     int first = counter.next();
10     int second = counter.next();
11
12     assertEquals( first + 1, second );
13   }
14
15   @Test
16   public void lowerBound() {
17     NumberRangeCounter counter = new NumberRangeCounter( LOWER_BOUND );
18
19     int actual = counter.next();
20
21     assertEquals( LOWER_BOUND, actual );
22   }
23 }
```

Test Structure

Setup Patterns

Inline Setup

```
01 public class NumberRangeCounterTest {
02
03   private static final int LOWER_BOUND = 1000;
04
05   @Test
06   public void subsequentNumber() {
07     NumberRangeCounter counter = new NumberRangeCounter( LOWER_BOUND );
08
09     int first = counter.next();
10     int second = counter.next();
11
12     assertEquals( first + 1, second );
13   }
14
15   @Test
16   public void lowerBound() {
17     NumberRangeCounter counter = new NumberRangeCounter( LOWER_BOUND );
18
19     int actual = counter.next();
20
21     assertEquals( LOWER_BOUND, actual );
22   }
23 }
```

Test Structure

Setup Patterns

Test Structure

Setup Patterns

```java
public class NumberRangeCounterTest {

  private static final int LOWER_BOUND = 1000;

  @Test
  public void subsequentNumber() {
    NumberRangeCounter counter = setUp();

    int first = counter.next();
    int second = counter.next();

    assertEquals( first + 1, second );
  }

  @Test
  public void lowerBound() {
    NumberRangeCounter counter = setUp();

    int actual = counter.next();

    assertEquals( LOWER_BOUND, actual );
  }

  private NumberRangeCounter setUp() {
    return new NumberRangeCounter( LOWER_BOUND );
  }
}
```

Test Structure

Setup Patterns

Delegate Setup

```
01 public class NumberRangeCounterTest {
02
03   private static final int LOWER_BOUND = 1000;
04
05   @Test
06   public void subsequentNumber() {
07     NumberRangeCounter counter = setUp();
08
09     int first = counter.next();
10     int second = counter.next();
11
12     assertEquals( first + 1, second );
13   }
14
15   @Test
16   public void lowerBound() {
17     NumberRangeCounter counter = setUp();
18
19     int actual = counter.next();
20
21     assertEquals( LOWER_BOUND, actual );
22   }
23
24   private NumberRangeCounter setUp() {
25     return new NumberRangeCounter( LOWER_BOUND );
26   }
27 }
```

Test Structure

Setup Patterns

Test Structure

Setup Patterns

```
01 public class NumberRangeCounterTest {
02
03   private static final int LOWER_BOUND = 1000;
04
05   private NumberRangeCounter counter;
06
07   @Before
08   public void setUp() {
09     counter = new NumberRangeCounter( LOWER_BOUND );
10   }
11
12   @Test
13   public void subsequentNumber() {
14     int first = counter.next();
15     int second = counter.next();
16
17     assertEquals( first + 1, second );
18   }
19
20   @Test
21   public void lowerBound() {
22     int actual = counter.next();
23
24     assertEquals( LOWER_BOUND, actual );
25   }
26 }
```

Test Structure

Setup Patterns

Implicit Setup

```
01 public class NumberRangeCounterTest {
02
03   private static final int LOWER_BOUND = 1000;
04
05   private NumberRangeCounter counter;
06
07   @Before
08   public void setUp() {
09     counter = new NumberRangeCounter( LOWER_BOUND );
10   }
11
12   @Test
13   public void subsequentNumber() {
14     int first = counter.next();
15     int second = counter.next();
16
17     assertEquals( first + 1, second );
18   }
19
20   @Test
21   public void lowerBound() {
22     int actual = counter.next();
23
24     assertEquals( LOWER_BOUND, actual );
25   }
26 }
```

Test Structure

Reusable Setup Helper

Object Mother

Test Fixture Registry

Implementation either as stateless test helper class or, in case the helper holds references to fixture or SUT objects, as stateful test helper object.

Test Structure

Implicit Teardown

```
1 @After
2 public void tearDown() {
3   counter.dispose();
4 }
```

Teardown is all about housekeeping and adds no information at all to a particular test

Test Structure

Corner Case Tests: Expected Exceptions

```
01 @Test
02 public void exeedsRange() {
03   NumberRangeCounter counter = new NumberRangeCounter( LOWER_BOUND, 0 );
04
05   try {
06     counter.next();
07     fail();
08   } catch( IllegalStateException expected ) {
09   }
10 }
```

Traditional approach: ugly try-catch block, mix of phases

Test Structure

Corner Case Tests: Expected Exceptions

Test Structure

Corner Case Tests: Expected Exceptions

```
1 @Test( expected = IllegalStateException.class )
2 public void exeedsRange() {
3   new NumberRangeCounter( LOWER_BOUND, ZERO_RANGE ).next();
4 }
```

Expected Annotation Method: might swallow setup problems, limited verification capabilities

Test Structure

Corner Case Tests: Expected Exceptions

Test Structure

Corner Case Tests: Expected Exceptions

```
01 public class NumberRangeCounterTest {
02
03   private static final int LOWER_BOUND = 1000;
04
05   @Rule
06   public ExpectedException thrown = ExpectedException.none();
07
08   @Test
09   public void exeedsRange() {
10     thrown.expect( IllegalStateException.class );
11
12     new NumberRangeCounter( LOWER_BOUND, 0 ).next();
13   }
14
15   [...]
16 }
```

ExpectedException Rule: Verification definition before execution phase

Test Structure

Corner Case Tests: Expected Exceptions

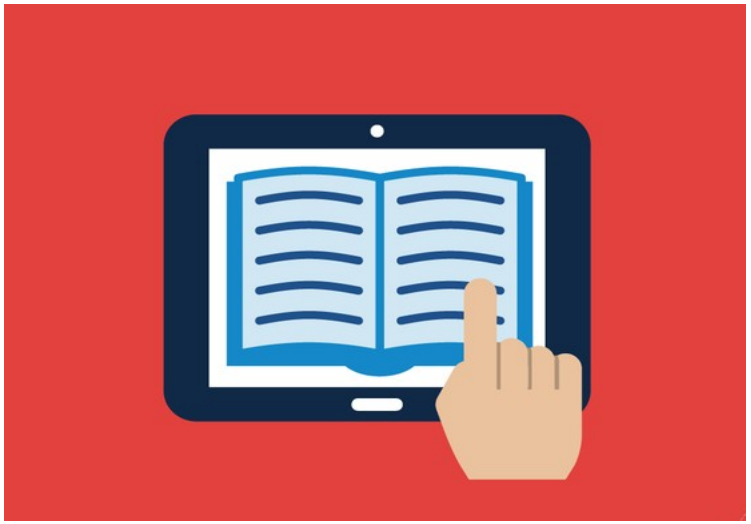
Test Structure

Corner Case Tests: Expected Exceptions

```
1 @Test
2 public void exeedsRange() {
3   NumberRangeCounter counter = new NumberRangeCounter( LOWER_BOUND, 0 );
4
5   Throwable actual = thrown( counter::next );
6
7   assertTrue( actual instanceof IllegalStateException );
8 }
```

Usage of a little execution utility and Java 8 Lambda expressions

Clean Unit Test Patterns



Why bother?

Structure

Isolation

Runners and Rules

Assertions

Q&A

Isolation

Dependencies: SUT and DOC

The tested Unit is usually referred to as system under test (SUT)

Components the SUT depends on are denoted as depended-on component (DOC)

Test related problems with DOCs

DOCs we cannot control might impede decent test verification

DOCs might also slow down test execution

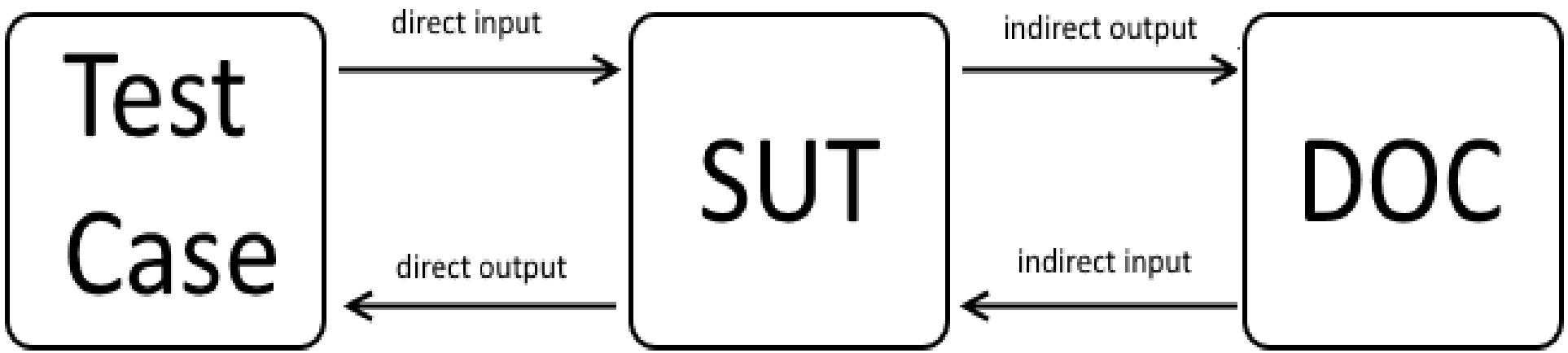
DOC's behavior may change unexpectedly e.g. due to the usage of a newer version of a third party library

Isolation

Isolation – A Unit Tester's SEP Field

Test concerns seperately and keep tests independent of each other!

Indirect Inputs and Outputs



Isolation

Test Double Patterns

A unit should be designed in a way that each DOC can be replaced by a so called Test Double, which is a lightweight stand-in component for the DOC.

A DOC is provided using dependency injection or a service locator. This ensures a loosely coupled micro-architecture that allows replacement of the DOC with the test double.

Isolation

Controlling Indirect Inputs with Stubs

```
1  public interface CounterStorage {
2    int getNumber();
3  }
```

```
01 public class CounterStorageDouble implements CounterStorage {
02
03   private int number;
04
05   public void setNumber( int number ) {
06     this.number = number;
07   }
08
09   @Override
10   public int getNumber() {
11     return number;
12   }
13 }
```

Isolation

Controlling Indirect Inputs with Stubs

Isolation

Controlling Indirect Inputs with Stubs

```
private CounterStorage storage;

@Before
public void setUp() {
  storage = new CounterStorageDouble();
  counter = new NumberRangeCounter( storage, LOWER_BOUND, RANGE );
}
```

Isolation

Controlling Indirect Inputs with Stubs

Isolation

Controlling Indirect Inputs with Stubs

```
01 private static final int IN_RANGE_NUMBER = LOWER_BOUND + RANGE / 2;
02
03 [...]
04
05 @Test
06 public void initialNumberFromStorage() {
07   storage.setNumber( IN_RANGE_NUMBER );
08
09   int actual = counter.next();
10
11   assertEquals( IN_RANGE_NUMBER, actual );
12 }
```

Isolation

Indirect Output Verification with Spies

```
1 public interface CounterStorage {
2   int getNumber();
3   void setNumber( int number );
4 }
```

The spy records the number value of the invocation in the test double's number field. This allows to verify the indirect output as shown here:

record

verify

```
1 @Test
2 public void storageOfStateChange() {
3   counter.next();
4
5   assertEquals( LOWER_BOUND + 1, storage.getNumber() );
6 }
```

Isolation

What About Mocks?

```
01 public class CounterStorageMock implements CounterStorage {
02
03   private int expectedNumber;
04   private boolean done;
05
06   public CounterStorageMock( int expectedNumber ) {
07     this.expectedNumber = expectedNumber;
08   }
09
10   @Override
11   public void setNumber( int actualNumber ) {
12     assertEquals( expectedNumber, actualNumber );
13     done = true;
14   }
15
16   public void verify() {
17     assertTrue( done );
18   }
19
20   @Override
21   public int getNumber() {
22     return 0;
23   }
24 }
```

Isolation

What About Mocks?

Behavior Verifcation

```
01 public class CounterStorageMock implements CounterStorage {
02
03   private int expectedNumber;
04   private boolean done;
05
06   public CounterStorageMock( int expectedNumber ) {
07     this.expectedNumber = expectedNumber;
08   }
09
10   @Override
11   public void setNumber( int actualNumber ) {
12     assertEquals( expectedNumber, actualNumber );
13     done = true;
14   }
15
16   public void verify() {
17     assertTrue( done );
18   }
19
20   @Override
21   public int getNumber() {
22     return 0;
23   }
24 }
```

Isolation

What About Mocks?

```
01 public class CounterStorageMock implements CounterStorage {
02
03   private int expectedNumber;
04   private boolean done;
05
06   public CounterStorageMock( int expectedNumber ) {
07     this.expectedNumber = expectedNumber;
08   }
09
10   @Override
11   public void setNumber( int actualNumber ) {
12     assertEquals( expectedNumber, actualNumber );
13     done = true;
14   }
15
16   public void verify() {
17     assertTrue( done );
18   }
19
20   @Override
21   public int getNumber() {
22     return 0;
23   }
24 }
```

Behavior Verifcation → line 12

Invocation Verification → line 17

Isolation

What About Mocks?

Isolation

What About Mocks?

```
01 @Test
02 public void storageOfStateChange() {
03   CounterStorageMock storage
04     = new CounterStorageMock( LOWER_BOUND + 1 );
05   NumberRangeCounter counter
06     = new NumberRangeCounter( storage, LOWER_BOUND, RANGE );
07
08   counter.next();
09
10   storage.verify();
11 }
```

Isolation

Spy or Mock?

Mocks break the usual test structure

Behavior verification is somewhat hidden

but

Mocks provide a precise stacktrace to the failure cause

Isolation

Test Double Frameworks

JMock, EasyMock mock based

Mockito spy based

```
01 @Test
02 public void storageOfStateChange() {
03   CounterStorage storage = mock( CounterStorage.class );
04   NumberRangeCounter counter
05     = new NumberRangeCounter( storage, LOWER_BOUND, RANGE );
06
07   counter.next();
08
09   verify( storage ).setNumber( LOWER_BOUND + 1 );
10 }
```

Isolation

Test Double Frameworks

> *If all you have is a hammer, everything looks like a nail*
>
> *Proverb*

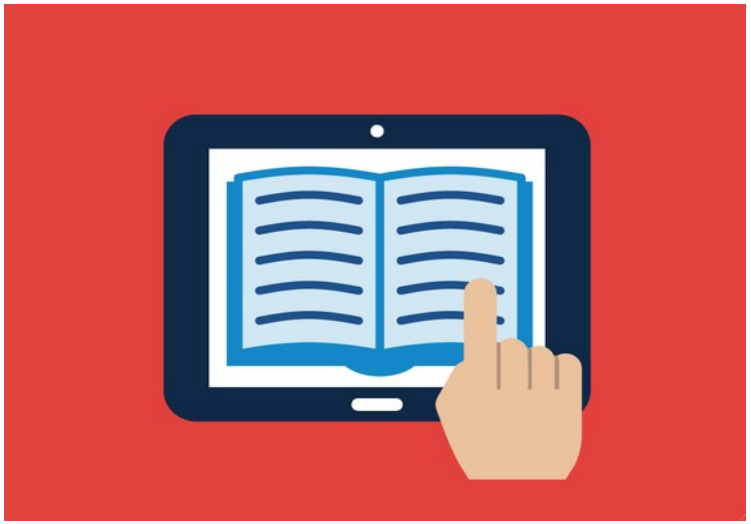
Only create test doubles for types you own

(indication for integration tests and an abstracting adapter layer)

A test double should not return another test double

(potential violation of law of demeter)

Clean Unit Test Patterns



Why bother?

Structure

Isolation

Runners and Rules

Assertions

Q&A

Runners

Suite and Categories

Use @RunWith to specify a particular test processor

```
1 @RunWith(Suite.class)
2 @SuiteClasses( {
3   NumberRangeCounterTest.class,
4   // list of test cases and other suites
5 } )
6 public class AllUnitTests {}
```

Runners

Suite and Categories

Runners

Suite and Categories

```
01 // definition of the available categories
02 public interface Unit {}
03 public interface Integration {}
04 public interface Acceptance {}
05
06 // category assignment of a test case
07 @Category(Unit.class)
08 public class NumberRangeCounterTest {
09   [...]
10 }
11
12 // suite definition that runs tests
13 // of the category 'Unit' only
14 @RunWith(Categories.class)
15 @IncludeCategory(Unit.class)
16 @SuiteClasses( {
17   NumberRangeCounterTest.class,
18   // list of test cases and other suites
19 } )
20 public class AllUnitTests {}
```

Runners

Parameterized Tests

Parameterized tests allow to run the same test against multiple data records provided as instance field(s) of the test case

fields

```
@Test
public void testConstructorParamValidation() {
  Throwable actual = thrown( () ->
    new NumberRangeCounter( storage, lowerBound, range ) );

  assertTrue( actual instanceof IllegalArgumentException );
  assertEquals( message, actual.getMessage() );
}
```

Runners

Parameterized Tests

Runners

Parameterized Tests

Specification of the Parameterized test processor using @RunWith

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

  private final String message;
  private final CounterStorage storage;
  private final int lowerBound;
  private final int range;
```

Runners

Parameterized Tests

Runners

Parameterized Tests

Field initialization takes place by constructor injection. Each data record is provided by a particular collector method annotated with @Parameters

data records ►

initializations ►

```
@Parameters
public static Collection<Object[]> data() {
  CounterStorage dummy = mock( CounterStorage.class );
  return Arrays.asList( new Object[][] {
    { NumberRangeCounter.ERR_PARAM_STORAGE_MISSING, null, 0, 0 },
    { NumberRangeCounter.ERR_LOWER_BOUND_NEGATIVE, dummy, -1, 0 },
     [...] // further data goes here...
  } );
}

public NumberRangeCounterTest(
  String message, CounterStorage storage, int lowerBound, int range )
{
  this.message = message;
  this.storage = storage;
  this.lowerBound = lowerBound;
  this.range = range;
}
```

Runners

JUnitParams

```
01 @RunWith( JUnitParamsRunner.class )
02 public class NumberRangeCounterTest {
03
04   public static Object data() {
05     CounterStorage dummy = mock( CounterStorage.class );
06     return $( $( ERR_PARAM_STORAGE_MISSING, null, 0, 0 ),
07               $( ERR_LOWER_BOUND_NEGATIVE, dummy, -1, 0 ) );
08   }
09
10   @Test
11   @Parameters( method = "data" )
12   public void testConstructorParamValidation(
13     String message, CounterStorage storage, int lowerBound, int range )
14   {
15     Throwable actual = thrown( () ->
16       new NumberRangeCounter( storage, lowerBound, range ) );
17
18     assertTrue( actual instanceof IllegalArgumentException );
19     assertEquals( message, actual.getMessage() );
20   }
21
22   [...]
23 }
```

Rules

What are JUnit Rules?

Rules provide a possibility to intercept test method calls similar as an AOP framework would do.

```
01 public class MyTest {
02
03   @Rule
04   public TemporaryFolder temporaryFolder = new TemporaryFolder();
05
06   @Test
07   public void testRun() throws IOException {
08     assertTrue( temporaryFolder.newFolder().exists() );
09   }
10 }
```

TemporaryFolder for example removes automatically all created files and directories after a test run.

A list of JUnit built-in Rules can be found at:
https://github.com/junit-team/junit/wiki/Rules

Rules

How does it work?

```
01 public class MyRule implements TestRule {
02
03   @Override
04   public Statement apply( Statement base, Description description ) {
05     return new MyStatement( base );
06   }
07 }
08
09 public class MyStatement extends Statement {
10
11   private final Statement base;
12
13   public MyStatement( Statement base ) {
14     this.base = base;
15   }
16
17   @Override
18   public void evaluate() throws Throwable {
19     System.out.println( "before" );
20     try {
21       base.evaluate();
22     } finally {
23       System.out.println( "after" );
24     }
25   }
26 }
```

Rules

How does it work?

Rules

How does it work?

```java
public class MyTest {

  @Rule
  public MyRule myRule = new MyRule();

  @Test
  public void testRun() {
    System.out.println( "during" );
  }
}
```

Test execution produces:

```
before
during
after
```

Rules

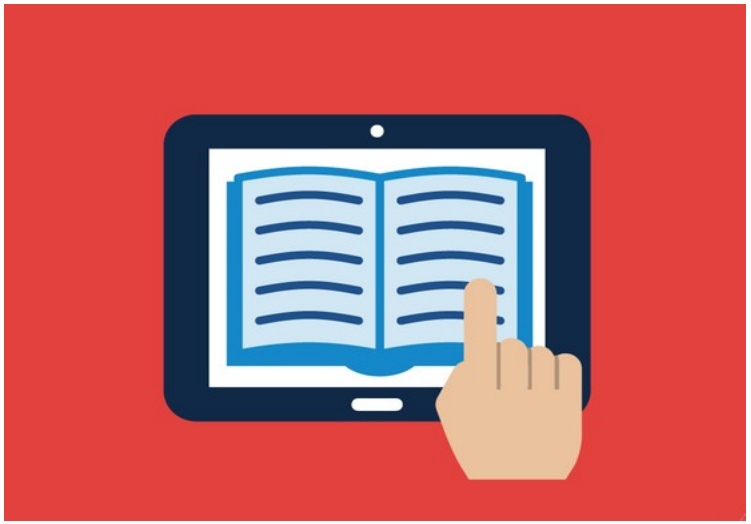
How does it work?

Rules

How does it work?

```
01 public class MyTest {
02
03   @Rule
04   public MyFixture myFixture = new MyFixture();
05
06   @Test
07   @Configuration( value = "configuration1" )
08   public void testRun1() {
09     // do some testing here
10   }
11
12   @Test
13   @Configuration( value = "configuration2" )
14   public void testRun2() {
15     // do some testing here
16   }
17 }
```

Clean Unit Test Patterns



Why bother?

Structure

Isolation

Runners and Rules

Assertions

Q&A

Assertions

JUnit Assert

The built-in assertion mechanism of JUnit is provided by the class org.junit.Assert:

```
fail();
fail( "Houston, We've Got a Problem." );

assertNull( actual );
assertNull( "Identifier must not be null.",
            actual );

assertTrue( counter.hasNext() );
assertTrue( "Counter should have a successor.",
            counter.hasNext() );

assertEquals( LOWER_BOUND, actual );
assertEquals( "Number should be lower bound value.",
              LOWER_BOUND,
              actual );
```

It is quite verbose and somewhat limited with respect to the expressiveness of assertions that require more complex predicates

Assertions

Hamcrest

A third-party library that claims to provide an API for creating flexible expressions of intent is Hamcrest:

```
assertThat( actual, is( equalTo( IN_RANGE_NUMBER ) ) );
```

MatcherAssert.assertThat(...) evaluates the execution result (actual) against a predicate (matcher-expression)

```
assertThat( "Actual number must not be equals to lower bound value.",
            actual,
            is( not( equalTo( LOWER_BOUND ) ) ) );
```

MatcherAssert provides an overloaded assertThat method for failure message specification

Assertions

Hamcrest

Assertions

Hamcrest

JUnit

Finished after 0,085 seconds

Runs: 1/1 Errors: 0 Failures: 1

increment [Runner: JUnit 4] (0,000 s)

Failure Trace

```
java.lang.AssertionError: Actual number must not be equals to lower bound value.
Expected: is not <1500>
     but: was <1500>
at org.hamcrest.MatcherAssert.assertThat(MatcherAssert.java:20)
at test.NumberRangeCounterTest.increment(NumberRangeCounterTest.java:92)
```

Assertions

AssertJ

The library AssertJ strives to improves verification by providing fluent assertions API:

```
Throwable actual = ...

assertThat( actual )
  .isInstanceOf( IllegalArgumentException.class )
  .hasMessage( EXPECTED_ERROR_MESSAGE );
```

Assertions.assertThat(...) verifies the execution result (actual) against fluently added conditions

```
Throwable actual = ...

assertThat( actual )
  .describedAs( "Expected exception does not match specification." )
  .hasMessage( EXPECTED_ERROR_MESSAGE )
  .isInstanceOf( NullPointerException.class );
```

The Assert instance provides the method describeAs(String) to specify a particular failure message

Assertions

AssertJ

Assertions

AssertJ

JUnit

Finished after 0,125 seconds

Runs: 8/8 Errors: 0 Failures: 3

testConstructorParamValidation (0,039 s)

[0] Parameter 'counterStorage' must not be null., null, 0, 0 (testConstructorParamValidation) (0,033 s)

[1] Parameter 'lowerBound' must not be negative., Mock for CounterStorage, hashCode: 866191240, -1, 0

Failure Trace

```
java.lang.AssertionError: [Expected exception does not match specification.]
Expecting:
 <java.lang.IllegalArgumentException: Parameter 'counterStorage' must not be null.>
to be an instance of:
 <java.lang.NullPointerException>
but was instance of:
 <java.lang.IllegalArgumentException>
at test.NumberRangeCounterTest.testConstructorParamValidation(NumberRangeCounterTest.java:38)
```

Assertions

Which one to use?

JUnit Assert is surely somewhat dated and less object-oriented

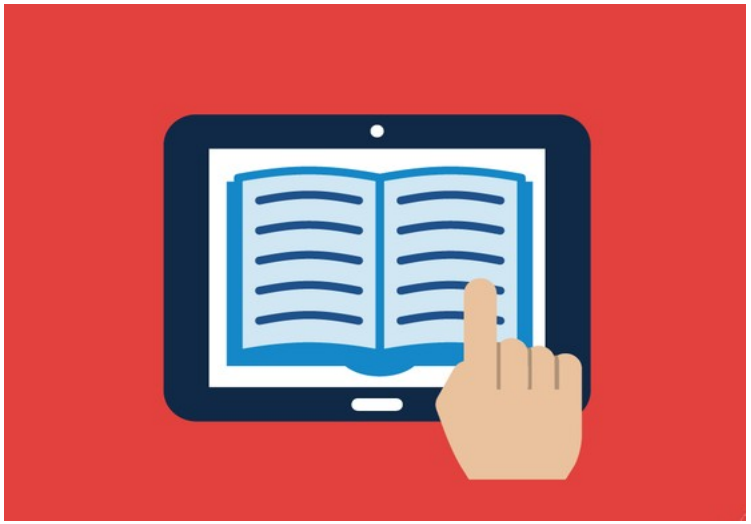
Hamcrest matchers provide a clean separation of assertion and predicate definition

AssertJ assertions score with a compact and easy to use programming style

Hamcrest and AssertJ support custom matchers/assertions for domain specific types

So now you are spoilt for choice…

Clean Unit Test Patterns



Why bother?

Structure

Isolation

Runners and Rules

Assertions

Q&A

Clean Unit Test Patterns

References

xUnit Test Patterns, Gerard Meszaros, 2007

Clean Code, Chapter 9: Unit Tests, Robert C. Martin, 2009

Growing Object-Oriented Software, Guided by Tests, Chapter 8, Steve Freeman, Nat Pryce, 2010

Practical Unit Testing with JUnit and Mockito, Appendix C. Test Spy vs. Mock, Tomek Kaczanowski, 2013

JUnit in a Nutshell: Yet Another JUnit Tutorial, http://www.codeaffine.com/2014/09/24/junit-nutshell-junit-tutorial, Frank Appel 2014

Clean JUnit Throwable-Tests with Java 8 Lambdas, http://www.codeaffine.com/2014/07/28/clean-junit-throwable-tests-with-java-8-lambdas/, Frank Appel 2014

Frank Appel

Blog: www.codeaffine.com
Email: fappel@codeaffine.com
Twitter: @frank_appel