



Unit Testing

The hard parts

October 10th, 2015

Shaun Abram

Blog: shaunabram.com
Email: shaun@abram.com
Twitter: [@shaunabram](https://twitter.com/shaunabram)
LinkedIn: linkedin.com/in/sabram

Test Obsessed?

How much do you know...

- Test coverage
- Dependency Injection
- Mock vs stubs
- Testing private or static methods
- Test driven development

What is a unit test?

A piece of code that executes a specific functionality ('unit') in the code, and

- Confirms behavior or result is as expected
- Determines if code is 'fit for use'
- Does it do what the developer intended?

Before, we manually verified, but not easily repeatable.

Why unit test?

Economics \$

but also...

- Drive design
- Create defensive code against bad input
- Act as safety buffers for regression bugs
- Provide documentation
- Clean code
- Less bugs

What is a unit?

- A class? A method? A single path through a method?
- The smallest testable part of an application.
- A single functional use case.

What makes a good unit test?

- Provides benefit!
- Readable, Understandable, Maintainable
- Independent
 - run in any order, no DB or File access
- Consistent / deterministic
- Runs fast
- Tests a single logical concept in the system

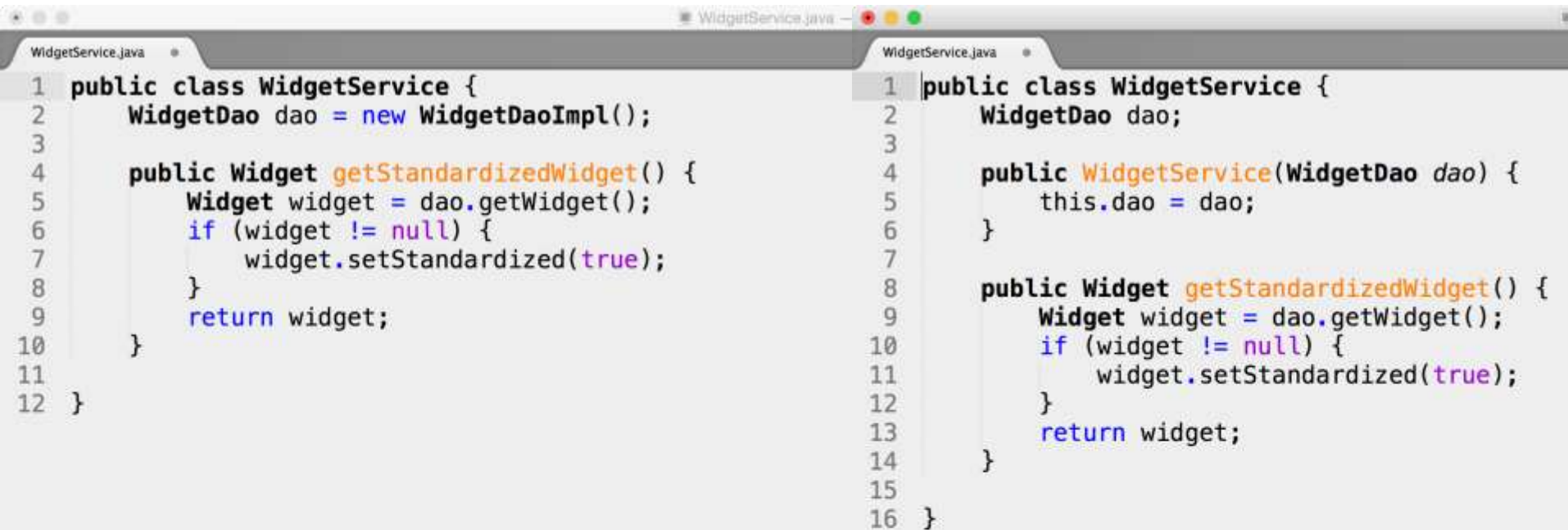
Unit testing limitations

- Can not prove the absence of bugs
- Lot's of code (x3-5)
- Some things difficult to test

Dependency Injection

WidgetService.java

```
1 public class WidgetService {  
2     WidgetDao dao = new WidgetDaoImpl();  
3  
4     public Widget getStandardizedWidget() {  
5         Widget widget = dao.getWidget();  
6         if (widget != null) {  
7             widget.setStandardized(true);  
8         }  
9         return widget;  
10    }  
11  
12 }
```



```
WidgetService.java
1 public class WidgetService {
2     WidgetDao dao = new WidgetDaoImpl();
3
4     public Widget getStandardizedWidget() {
5         Widget widget = dao.getWidget();
6         if (widget != null) {
7             widget.setStandardized(true);
8         }
9         return widget;
10    }
11
12 }
```

```
WidgetService.java
1 public class WidgetService {
2     WidgetDao dao;
3
4     public WidgetService(WidgetDao dao) {
5         this.dao = dao;
6     }
7
8     public Widget getStandardizedWidget() {
9         Widget widget = dao.getWidget();
10        if (widget != null) {
11            widget.setStandardized(true);
12        }
13        return widget;
14    }
15
16 }
```

Dependency Injection

- Useful for testing – inject test doubles
- Also helps reduce coupling

Test Doubles

Mocks

Stubs

Dummies

Spies

Fakes

Mocks

Stubs

Dummies

Spies

Fakes

Test Doubles: Mocks

Mocks: An overloaded term!

- uses behavior verification
- objects pre-programmed with expectations

WidgetService.java

```
1 public class WidgetService {
2     final WidgetDao dao;
3
4     public WidgetService(WidgetDao dao) {
5         this.dao = dao;
6     }
7
8     public void createWidget(Widget widget) {
9         //misc business logic
10        //e.g. validating widget is valid
11
12        dao.saveWidget(widget);
13    }
14
15 }
```

```
WidgetService.java
1 public class WidgetService {
2     final WidgetDao dao;
3
4     public WidgetService(WidgetDao dao) {
5         this.dao = dao;
6     }
7
8     public void createWidget(Widget widget) {
9         //misc business logic
10        //e.g. validating widget is valid
11
12        dao.saveWidget(widget);
13    }
14
15 }

WidgetServiceTest.java
1 public class WidgetServiceTest {
2
3     //test fixtures
4     WidgetDao widgetDao = mock(WidgetDao.class);
5     WidgetService widgetService
6         = new WidgetService(widgetDao);
7     Widget widget = new Widget();
8
9     @Test
10    public void createWidget_saves_widget() {
11        //call method under test
12        widgetService.createWidget(widget);
13
14        //verify expectation
15        verify(widgetDao).saveWidget(widget);
16    }
17
18 }
```

- Mocks use behavior verification
- Can specify how to respond when called
- Roll your own?

Test Doubles

Mocks

Stubs

Dummies

Spies

Fakes

Test Doubles: Stubs

‘stubs out’
or provides a simplified version
of the implementation
for the purposes of testing

WidgetService.java

```
1 public class WidgetService {  
2     final ManagerService manager;  
3  
4     public WidgetService(ManagerService m) {  
5         this.manager = m;  
6     }  
7  
8     public void standardize(Widget widget) {  
9         if (manager.isActive()) {  
10             widget.setStandardized(true);  
11         }  
12     }  
13 }
```

```
WidgetService.java
1 public class WidgetService {
2     final ManagerService manager;
3
4     public WidgetService(ManagerService m) {
5         this.manager = m;
6     }
7
8     public void standardize(Widget widget) {
9         if (manager.isActive()) {
10             widget.setStandardized(true);
11         }
12     }
13 }

WidgetServiceTest.java
1 public class WidgetServiceTest {
2
3     class ManagerServiceStub extends ManagerService {
4         @Override
5         public boolean isActive() {
6             return true;
7         }
8     }
9
10    ManagerServiceStub managerServiceStub
11        = new ManagerServiceStub();
12    WidgetService widgetService
13        = new WidgetService(managerServiceStub);
14    Widget widget = new Widget();
15
16    @Test
17    public void standardize_widget_when_active() {
18        //call method under test
19        widgetService.standardize(widget);
20
21        //verify state
22        assertTrue(widget.isStandardized());
23    }
24 }
```

- Stubs can use state or behavior verification
- Provides a useful approach for test fixture configurability

Test Doubles

Mocks

Stubs

Dummies

Spies

Fakes

Test Doubles: Dummies

- A very dumb class!
- Contains next to nothing - enough to compile
- Pass when you don't expect to be used

```
1 public class WidgetDaoDummy implements WidgetDao {
2
3     @Override
4     public Widget getWidget() {
5         throw new RuntimeException("Not expected to be called");
6     }
7
8     @Override
9     public void saveWidget(Widget widget) {
10        throw new RuntimeException("Not expected to be called");
11    }
12
13 }
```

- Use dummies with state or behavior verification
- Can create as inner class
- Can replicate with mocks

Unit Testing – the tricky parts

- Legacy code
- Privates
- Statics

General approach to testing legacy code

- 1) Start with coarse grained tests
 - No modifications
 - Strict and rigid
 - Detect regressions
 - Short term bridges - delete

General approach to testing legacy code

- 1) Start with coarse grained tests
- 2) Add finer grained tests
 - Incrementally add more unit
 - start gently refactoring
 - Use TDD – if possible
 - Test from 1) should protect you

General approach to testing legacy code

- 1) Start with coarse grained tests
- 2) Add finer grained tests
- 3) Continuously refactor

Use patterns such as

- Extract Method/Class
- Move Method/Field
- tease apart methods (>10 lines smells)

Unit Testing – the tricky parts

- Legacy code
- **Privates**
- Statics

How do you test private methods?

Indirectly! Best tested via public interface

But sometimes...

- Legacy code
 - With limited capability to refactor, or
 - adding a safety net before refactoring
- A public method calls several private methods, each with complex logic

Testing Private Methods

1. Refactor
2. Change the visibility
 - Bad practice
 - Public API? Very bad practice!
 - Internal & stepping stone -> lesser evil
 - Java private -> package; .Net protected or internal?
 - Document (`@VisibleForTesting`)

Testing Private Methods

1. Refactor
2. Change the visibility
3. Use Reflection
 - No code modification, but...
4. Other options...
 - Testing frameworks
 - `InternalsVisibleToAttribute`
 - Private Accessors

Testing static methods

untitled

```
1 public class WidgetService {  
2  
3     public Widget getStandardizedWidget() {  
4         Widget widget = WidgetDao.getWidget();  
5         if (widget != null) {  
6             widget.setStandardized(true);  
7         }  
8         return widget;  
9     }  
10  
11 }
```



```
1 public class WidgetService {  
2  
3     public Widget getStandardizedWidget() {  
4         Widget widget = WidgetDao.getWidget();  
5         if (widget != null) {  
6             widget.setStandardized(true);  
7         }  
8         return widget;  
9     }  
10  
11 }
```

```
1 public class WidgetService {  
2     final WidgetDao dao;  
3  
4     public WidgetService(WidgetDao dao) {  
5         this.dao = dao;  
6     }  
7  
8     public Widget getStandardizedWidget()  
9     {  
10         Widget widget = dao.getWidget();  
11         if (widget != null) {  
12             widget.setStandardized(true);  
13         }  
14         return widget;  
15     }  
16 }
```

```
1 public class WidgetService {
2
3     public Widget getStandardizedWidget() {
4         Widget widget = WidgetDao.getWidget();
5         if (widget != null) {
6             widget.setStandardized(true);
7         }
8         return widget;
9     }
10
11 }
```

```
1 public class WidgetService {
2     final WidgetDao dao;
3
4     public WidgetService(WidgetDao dao) {
5         this.dao = dao;
6     }
7
8     public Widget getStandardizedWidget() {
9         Widget widget = dao.getWidget();
10        if (widget != null) {
11            widget.setStandardized(true);
12        }
13        return widget;
14    }
15 }
```

```
1 public class WidgetServiceTest {
2
3     WidgetDao widgetDaoMock= mock(WidgetDao.class);
4     WidgetService widgetService = new WidgetService(widgetDaoMock);
5     Widget unstandardizedWidget = new Widget();
6
7     @Test
8     public void getStandardizedWidget_returns_standardized_widget() {
9         //set expectations on the mock
10        when(widgetDaoMock.getWidget()).thenReturn(unstandardizedWidget);
11        //call the method under test
12        Widget widget = widgetService.getStandardizedWidget();
13        //verify behavior
14        assertTrue(widget.isStandardized());
15    }
16
17 }
```

Testing static methods

- DI ideal
- But sometimes not pragmatic

untitled

```
1 public class WidgetService {  
2  
3     public Widget getStandardizedWidget() {  
4         Widget widget = WidgetDao.getWidget();  
5         if (widget != null) {  
6             widget.setStandardized(true);  
7         }  
8         return widget;  
9     }  
10  
11 }
```

```
untitled
1 public class WidgetService {
2
3     public Widget getStandardizedWidget() {
4         Widget widget = WidgetDao.getWidget();
5         if (widget != null) {
6             widget.setStandardized(true);
7         }
8         return widget;
9     }
10
11 }
```

```
untitled
1 public class WidgetService {
2
3     public Widget getStandardizedWidget() {
4         Widget widget = getWidget();
5         if (widget != null) {
6             widget.setStandardized(true);
7         }
8         return widget;
9     }
10
11     Widget getWidget() {
12         return WidgetDao.getWidget();
13     }
14 }
```

Refactor to wrap the static call in an instance method
Which can then be mocked...

```
untitled
1 public class WidgetService {
2
3     public Widget getStandardizedWidget() {
4         Widget widget = WidgetDao.getWidget();
5         if (widget != null) {
6             widget.setStandardized(true);
7         }
8         return widget;
9     }
10
11 }
```

```
untitled
1 public class WidgetService {
2
3     public Widget getStandardizedWidget() {
4         Widget widget = getWidget();
5         if (widget != null) {
6             widget.setStandardized(true);
7         }
8         return widget;
9     }
10
11     Widget getWidget() {
12         return WidgetDao.getWidget();
13     }
14 }
```

```
WidgetServiceTest.java
1 public class WidgetServiceTest {
2
3     WidgetService service = partialMock(new WidgetService());
4     Widget unstandardizedWidget = new Widget();
5
6     @Test
7     public void getStandardizedWidget_returns_standardized_widget() {
8         //set expectations
9         when(service.getWidget()).thenReturn(unstandardizedWidget);
10        //cal method under test
11        Widget widget = service.getStandardizedWidget();
12        //verify behavior
13        assertTrue(widget.isStandardized());
14    }
15 }
16
```


WidgetServiceTest.java

```
1 public class WidgetServiceTest {
2
3     WidgetService service = partialMock(new WidgetService());
4     Widget unstandardizedWidget = new Widget();
5
6     @Test
7     public void getStandardizedWidget_returns_standardized_widget() {
8         //set expectations
9         when(service.getWidget()).thenReturn(unstandardizedWidget);
10        //cal method under test
11        Widget widget = service.getStandardizedWidget();
12        //verify behavior
13        assertTrue(widget.isStandardized());
14    }
15 }
16
```

WidgetServiceTest.java

```
1 public class WidgetServiceTest {
2
3     WidgetService service = partialMock(new WidgetService());
4     Widget unstandardizedWidget = new Widget();
5
6     @Test
7     public void getStandardizedWidget_returns_standardized_widget() {
8         //set expectations
9         when(service.getWidget()).thenReturn(unstandardizedWidget);
10        //call method under test
11        Widget widget = service.getStandardizedWidget();
12        //verify behavior
13        assertTrue(widget.isStandardized());
14    }
15 }
```

WidgetServiceStubTest.java

UNREGISTERED

WidgetServiceStubTest.java

```
1 public class WidgetServiceStubTest {
2     WidgetServiceStub service = new WidgetServiceStub();
3     Widget unstandardizedWidget = new Widget();
4
5     class WidgetServiceStub extends WidgetService {
6         public Widget getStandardizedWidget() {
7             return unstandardizedWidget;
8         }
9     }
10
11     @Test
12     public void getStandardizedWidget_returns_standardized_widget() {
13         //call method under test
14         Widget widget = service.getStandardizedWidget();
15         //verify results
16         assertTrue(widget.isStandardized());
17     }
18 }
```


Testing static methods

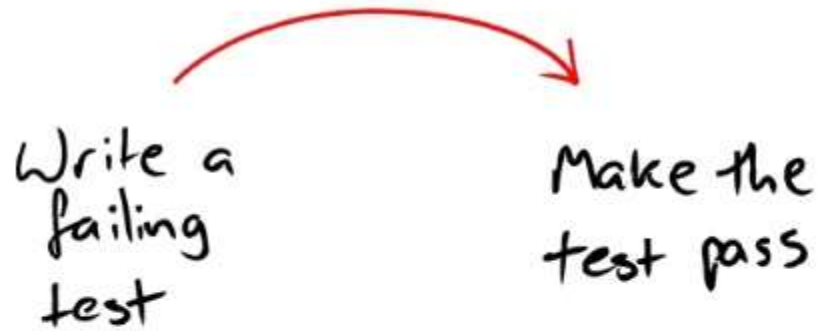
- Refactor – use DI
- Wrap static call in a instance method
- Use a mocking framework?

Test Driven Development

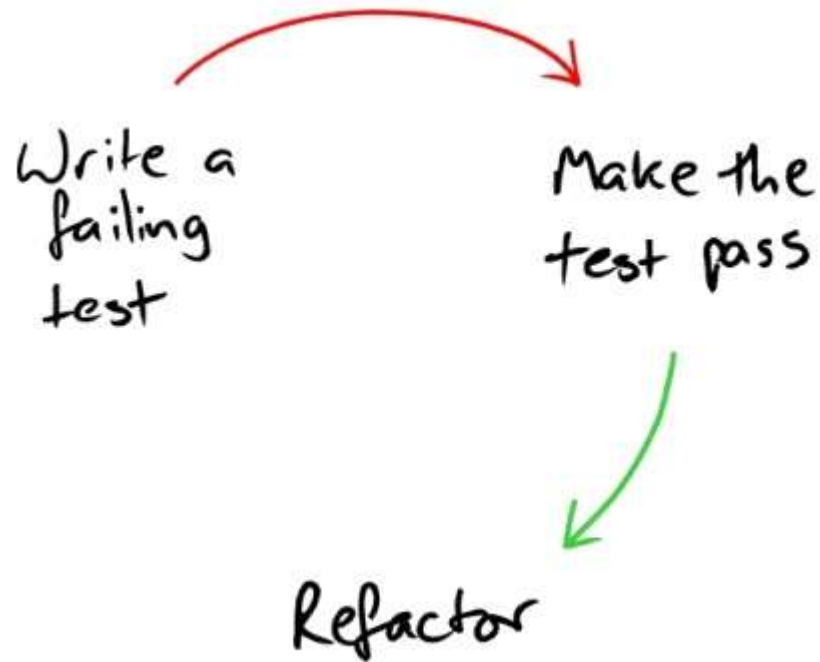
Test Driven Development

Write a
failing
test

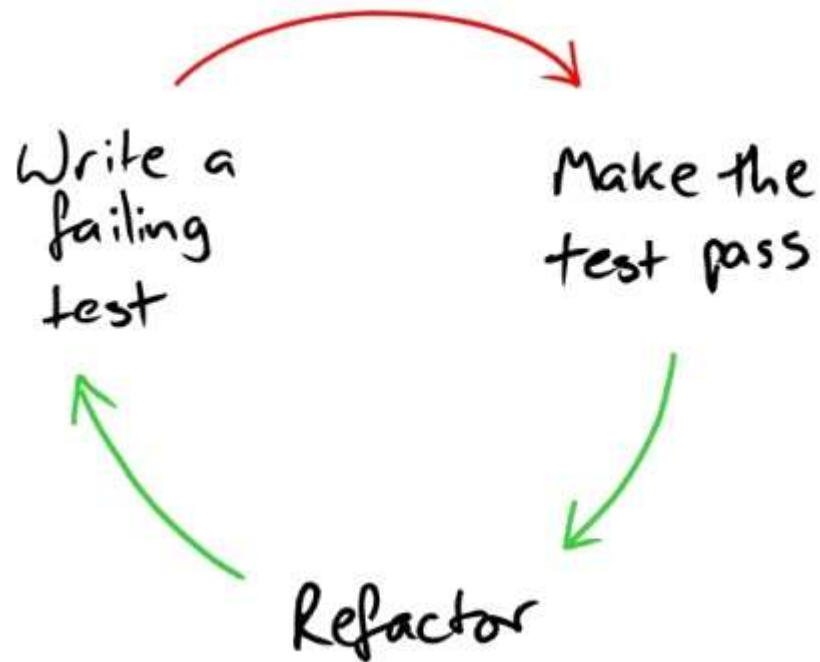
Test Driven Development



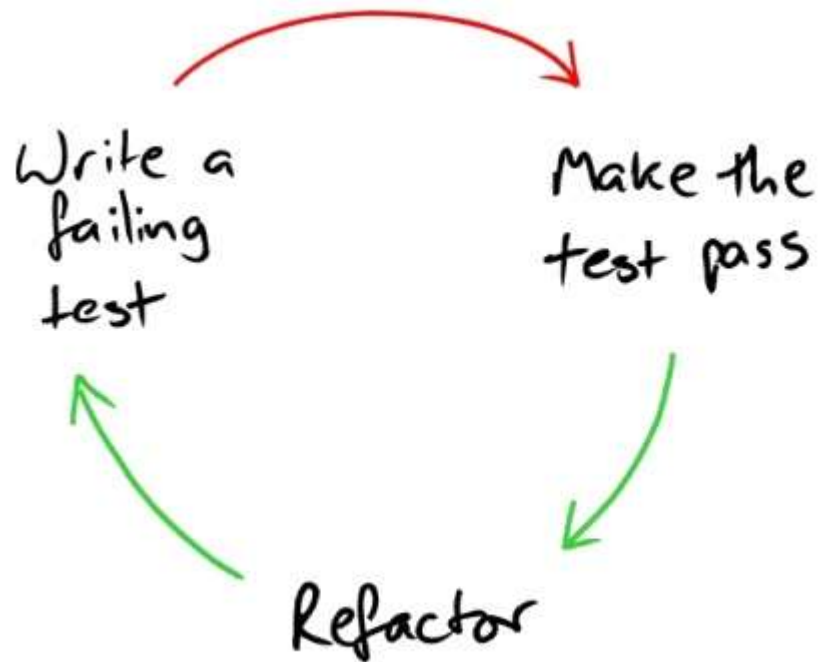
Test Driven Development



Test Driven Development



Test Driven Development



Red - Green - Refactor: the TDD Mantra
No new functionality without a failing test
No refactoring without passing tests

Test Driven Development Example...

Test Driven Development Example...

Create a **StringCalculator** class
with an **add** method
which takes a comma separate String of numbers
and returns their sum.

Example input	Result
	0
1	1
2	2
1,2	3
1,2,100	103

StringCalculatorTest.java

```
1
2
3 import org.junit.Test;
4
5 public class StringCalculatorTest {
6
7     @Test
8     public void add_returns_0_for_empty_String() {
9
10 }
11
12 }
```

4: Run 5: TODO

Tests Passed: 2 passed (6 minutes ago)

9:9 UTF-8 Insert 158M of 494M

Commander Ant Build Data Sources JetBrains IDEtalk Maven Projects Z: Structure

2: Favorites

Event Log

1
2
3
4
5
6
7
8
9
10
11
12
13

StringCalculatorTest.java

```
import org.junit.Test;

public class StringCalculatorTest {

    @Test
    public void add_returns_0_for_empty_String() {
        StringCalculator calculator = new StringCalculator();
        calculator.add("");
    }
}
```

2: Favorites

Commander
Ant Build
Data Sources
JetGradle
IDEtalk
Maven Projects
Z: Structure

4: Run
6: TODO
Event Log

Tests Passed: 2 passed (9 minutes ago)
10:26
UTF-8
Insert
170M of 494M

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

StringCalculatorTest.java

```
import ...

public class StringCalculatorTest {

    @Test
    public void add_returns_0_for_empty_String() {
        StringCalculator calculator = new StringCalculator();
        assertEquals(calculator.add(""), 0);
    }
}
```

1: Project

2: Favorites

Commander

Ant Build

Data Sources

JetCraide

IDEtalk

Maven Projects

Z: Structure

Run

TODO

Event Log

Tests Passed: 2 passed (4 minutes ago)

15:2

UTF-8

Insert

121M of 494M

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

StringCalculatorTest.java

```
1  
2  
3 import ...  
4  
5  
6  
7 public class StringCalculatorTest {  
8  
9     @Test  
10    public void add_returns_0_for_empty_String() {  
11        StringCalculator calculator = new StringCalculator();  
12        assertThat(calculator.add(""), isEqualTo(0));  
13    }  
14  
15 }
```

Project
Favorites

1
2
3
4
5
6
7
8
9

StringCalculator.java

```
1  
2  
3 public class StringCalculator {  
4  
5     public Integer add(String numString) {  
6         return null;  
7     }  
8  
9 }
```

Commander
Ant Build
Data Sources
JetGradle
IDEvark
Maven Projects
Z: Structure

4: Run
6: TODO

Event Log

Tests Failed: 0 passed, 1 failed (moments ago)

7:29
UTF-8
Insert

132M of 494M

Project

StringCalculatorTest.java

1
2
3 import ...
6
7 public class StringCalculatorTest {
8
9 @Test
10 public void add_returns_0_for_empty_String() {
11 StringCalculator calculator = new StringCalculator();
12 assertThat(calculator.add("")).isEqualTo(0);
13 }
14
15 }

StringCalculator.java

1
2
3 public class StringCalculator {
4
5 public Integer add(String numString) {
6 return null;
7 }
8
9 }

Run StringCalculatorTest

Done: 1 of 1 Failed: 1 (0.028 s)

StringCalculatorTest (com.blackrock.tar.testing)
└─ add_returns_0_for_empty_String

"C:\Program Files\Java\jdk1.7.0_11\bin\java" -ea -Didea.launcher.port=7534 "-Didea.launcher.port=7534"
java.lang.AssertionError: expecting actual not to be null
at com.blackrock.tar.testing.StringCalculatorTest.add_returns_0_for_empty_String(Str

4: Run 6: TODO

Tests Failed: 0 passed, 1 failed (a minute ago)

7:29 UTF-8 Insert 114M of 494M

Commander
Ant Build
Data Sources
JetGradle
IDEtalk
Maven Projects
Z: Structure

1: Project

StringCalculatorTest.java

```
1
2
3 import ...
4
5
6 public class StringCalculatorTest {
7
8     @Test
9     public void add_returns_0_for_empty_String() {
10         StringCalculator calculator = new StringCalculator();
11         assertThat(calculator.add("")).isEqualTo(0);
12     }
13
14     @Test
15     public void add_returns_integer_value_of_single_number_String() {
16         StringCalculator calculator = new StringCalculator();
17         assertThat(calculator.add("1")).isEqualTo(1);
18     }
19 }
20
21
```

2: Favorites

StringCalculator.java

```
1
2
3 public class StringCalculator {
4
5     public Integer add(String numString) {
6         return 0;
7     }
8 }
9
```

Commander
Ant Build
Data Sources
JetCade
IDEtalk
Maven Projects
Z: Structure

4: Run 6: TODO

Event Log

Tests Passed: 1 passed (6 minutes ago)

9:1 UTF-8 Insert 169M of 494M

The screenshot shows an IDE with two open Java files. The left file, `StringCalculatorTest.java`, contains two JUnit tests. The right file, `StringCalculator.java`, contains a single method `add` that always returns 0. The bottom panel shows the test results for `StringCalculatorTest`, indicating that one test failed. The error message is:

```
org.junit.ComparisonFailure:
Expected :1
Actual   :0
```

1: Project

2: Favorites

StringCalculatorTest.java

```
1
2
3 import ...
4
5
6 public class StringCalculatorTest {
7
8     @Test
9     public void add_returns_0_for_empty_String() {
10         StringCalculator calculator = new StringCalculator();
11         assertThat(calculator.add(""), isEqualTo(0));
12     }
13
14     @Test
15     public void add_returns_integer_value_of_single_number_String() {
16         StringCalculator calculator = new StringCalculator();
17         assertThat(calculator.add("1"), isEqualTo(1));
18     }
19 }
20
21
```

StringCalculator.java

```
1
2
3 public class StringCalculator {
4
5     public Integer add(String numString) {
6         if (numString.isEmpty()) return 0;
7         return 1;
8     }
9
10
```

4: Run

6: TODO

Event Log

Tests Failed: 1 passed, 1 failed (2 minutes ago)

7:17

UTF-8

Insert

115M of 494M

Commander

Ant Build

Data Sources

JetCradle

IDEvark

Maven Projects

Z: Structure

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21

```
StringCalculatorTest.java
import ...

public class StringCalculatorTest {

    @Test
    public void add_returns_0_for_empty_String() {
        StringCalculator calculator = new StringCalculator();
        assertThat(calculator.add(""), isEqualTo(0));
    }

    @Test
    public void add_returns_integer_value_of_single_number_String() {
        StringCalculator calculator = new StringCalculator();
        assertThat(calculator.add("1"), isEqualTo(1));
    }
}
```

1
2
3
4
5
6
7
8
9
10

```
StringCalculator.java
public class StringCalculator {

    public Integer add(String numString) {
        if (numString.isEmpty()) return 0;
        return 1;
    }
}
```

Run StringCalculatorTest

Done: 2 of 2 (0.02 s)

StringCalculatorTest (com.blackrock.tar.testing)
├─ add_returns_integer_value_of_single_numb
└─ add_returns_0_for_empty_String

"C:\Program Files\Java\jdk1.7.0_11\bin\java" -ea -Didea.launcher.port=7533 "-Didea.launcher.port=7533"
Process finished with exit code 0

4: Run

6: TODO

Tests Passed: 2 passed (moments ago)

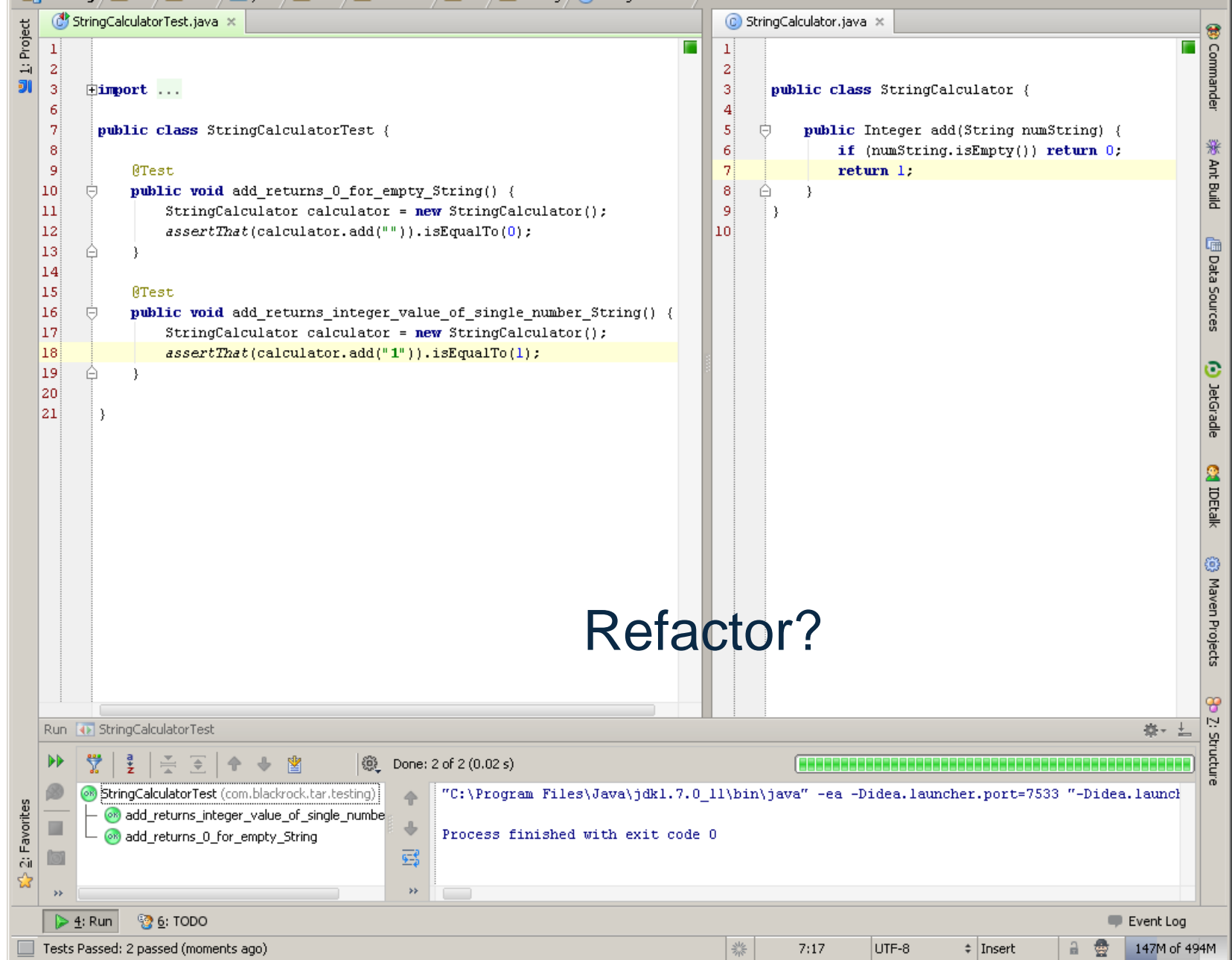
7:17

UTF-8

Insert

Event Log

147M of 494M



1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27

StringCalculatorTest.java

```
1  
2  
3  import ...  
4  
5  
6  
7  
8  public class StringCalculatorTest {  
9  
10     StringCalculator calculator;  
11  
12     @Before  
13     public void setup() {  
14         calculator = new StringCalculator();  
15     }  
16  
17     @Test  
18     public void add_returns_0_for_empty_String() {  
19         assertThat(calculator.add("")).isEqualTo(0);  
20     }  
21  
22     @Test  
23     public void add_returns_integer_value_of_single_number_String() {  
24         assertThat(calculator.add("1")).isEqualTo(1);  
25     }  
26  
27 }
```

2: Favorites

1
2
3
4
5
6
7
8
9
10

StringCalculator.java

```
1  
2  
3  public class StringCalculator {  
4  
5      public Integer add(String numString) {  
6          if (numString.isEmpty()) return 0;  
7          return 1;  
8      }  
9  
10 }
```

Commander
Ant Build
Data Sources
JatGracie
IDEvark
Maven Projects
Z: Structure

Run
TODO

Tests Passed: 2 passed (4 minutes ago)

23:70
UTF-8
Insert
172M of 494M

Event Log

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28

StringCalculatorTest.java

```
1  
2  
3 import ...  
4  
5  
6  
7  
8 public class StringCalculatorTest {  
9  
10     StringCalculator calculator;  
11  
12     @Before  
13     public void setup() {  
14         calculator = new StringCalculator();  
15     }  
16  
17     @Test  
18     public void add_returns_0_for_empty_String() {  
19         assertThat(calculator.add("")).isEqualTo(0);  
20     }  
21  
22     @Test  
23     public void add_returns_integer_value_of_single_number_String() {  
24         assertThat(calculator.add("1")).isEqualTo(1);  
25         assertThat(calculator.add("2")).isEqualTo(2);  
26     }  
27  
28 }
```

2: Favorites

1
2
3
4
5
6
7
8
9
10

StringCalculator.java

```
1  
2  
3 public class StringCalculator {  
4  
5     public Integer add(String numString) {  
6         if (numString.isEmpty()) return 0;  
7         return 1;  
8     }  
9  
10 }
```

Commander
Ant Build
Data Sources
JatGraddle
IDEvarkit
Maven Projects
Z: Structure

Run
TODO

Tests Passed: 2 passed (2 minutes ago)

25:37
UTF-8
Insert
103M of 494M

Event Log

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28

```
StringCalculatorTest.java
import ...

public class StringCalculatorTest {

    StringCalculator calculator;

    @Before
    public void setup() {
        calculator = new StringCalculator();
    }

    @Test
    public void add_returns_0_for_empty_String() {
        assertThat(calculator.add(""), isEqualTo(0));
    }

    @Test
    public void add_returns_integer_value_of_single_number_String() {
        assertThat(calculator.add("1"), isEqualTo(1));
        assertThat(calculator.add("2"), isEqualTo(2));
    }
}
```

1
2
3
4
5
6
7
8
9
10

```
StringCalculator.java
public class StringCalculator {

    public Integer add(String numString) {
        if (numString.isEmpty()) return 0;
        return Integer.parseInt(numString);
    }
}
```

Run StringCalculatorTest

StringCalculatorTest (com.blackrock.tar.testing)

add_returns_0_for_empty_String

add_returns_integer_value_of_single_numbe

Done: 2 of 2 (0.017 s)

"C:\Program Files\Java\jdk1.7.0_11\bin\java" -ea -Didea.launcher.port=7535 "-Didea.launcher.port=7535"

Process finished with exit code 0

4: Run 6: TODO

Tests Passed: 2 passed (moments ago)

7:42 UTF-8 Insert 99M of 494M

Commander
Ant Build
Data Sources
JetGradle
IDEvark
Maven Projects
Z: Structure

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35

StringCalculatorTest.java

```
import ...

public class StringCalculatorTest {

    StringCalculator calculator;

    @Before
    public void setup() {
        calculator = new StringCalculator();
    }

    @Test
    public void add_returns_0_for_empty_String() {
        checkAdd("", 0);
    }

    @Test
    public void add_returns_integer_value_of_single_number_String() {
        checkAdd("1", 1);
        checkAdd("1234", 1234);
    }

    @Test
    public void add_returns_integer_sum_of_multi_number_String() {
        checkAdd("1,2", 3);
    }

    private void checkAdd(String numString, int num) {
        assertThat(calculator.add(numString)).isEqualTo(num);
    }
}
```

StringCalculator.java

```
public class StringCalculator {

    public Integer add(String numString) {
        if (numString.isEmpty()) return 0;
        String[] numbers = numString.split(",");
        return sum(numbers);
    }

    private int sum(String[] numbers) {
        int sum = 0;
        for (String number : numbers) {
            sum += Integer.parseInt(number);
        }
        return sum;
    }
}
```

Run

StringCalculatorTest

Done: 3 of 3 (0.019 s)

StringCalculatorTest (com.blackrock.tar.testing)

add_returns_0_for_empty_String

add_returns_integer_value_of_single_number

add_returns_integer_sum_of_multi_number

"C:\Program Files\Java\jdk1.7.0_11\bin\java" -ea -Didea.launcher.port=7536 "-Didea.launcher.port=7536"

Process finished with exit code 0

4: Run

6: TODO

Event Log

Tests Passed: 3 passed (moments ago)

18:2

UTF-8

Insert

113M of 494M

The screenshot shows an IDE window with a Java file named `StringCalculatorTest.java`. The code is as follows:

```
1
2
3 import ...
4
5
6 public class StringCalculatorTest {
7
8     StringCalculator calculator;
9
10
11     @Before
12     public void setup() {
13         calculator = new StringCalculator();
14     }
15
16     @Test
17     public void add_returns_0_for_empty_String() {
18         checkAdd("", 0);
19     }
20
21     @Test
22     public void add_returns_integer_value_of_single_number_String() {
23         checkAdd("1", 1);
24         checkAdd("1234", 1234);
25     }
26
27     @Test
28     public void add_returns_integer_sum_of_multi_number_String() {
29         checkAdd("1,2", 3);
30     }
31
32     private void checkAdd(String numString, int num) {
33         assertThat(calculator.add(numString)).isEqualTo(num);
34     }
35 }
```

The right-hand pane shows the **Structure** view for the `StringCalculatorTest` class, listing the following methods:

- `setup():void`
- `add_returns_0_for_empty_String():void`
- `add_returns_integer_value_of_single_number_String():void`
- `add_returns_integer_sum_of_multi_number_String():void`

On the right side of the IDE, there is a red text overlay:

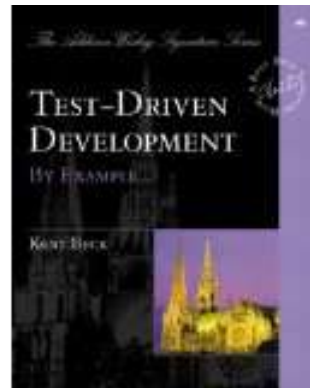
These tests act like the original developer looking over your shoulder and advising you, long after that developer has left...

The bottom status bar indicates that 3 tests passed 6 minutes ago, and the memory usage is 130M of 494M.

Recommended reading

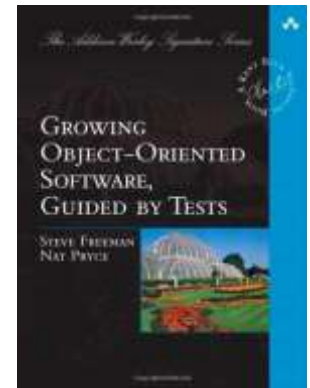
Test Driven Development

Kent Beck



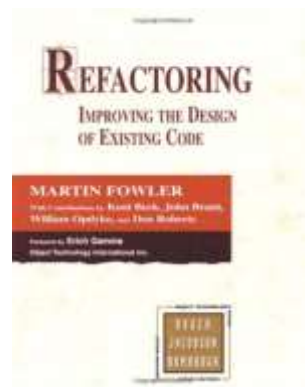
Growing Object-Oriented Software, Guided by Tests

Freeman & Pryce



Refactoring: Improving the Design of Existing Code

Martin Fowler, Kent Beck et. al.



Effective Unit Testing

Lasse Koskela



Recommended reading

Unit Test Best Practices -

<https://wiki.tlcinternal.com/display/TD/Unit+Test+Best+Practices>

Mocks Arent Stubs -

<http://www.martinfowler.com/articles/mocksArentStubs.html>

Questions?