Test Driven Development

Definitions

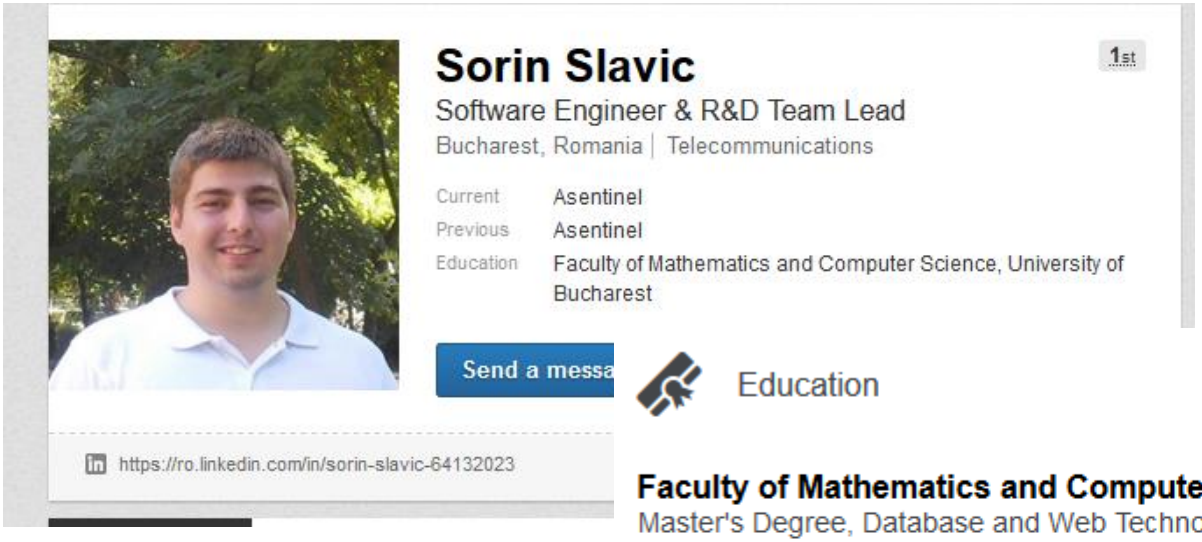
Example 1

Example 2

Example 3 (maybe)

Conclusions + Q&A

Me



Sorin Slavic 1st

Software Engineer & R&D Team Lead

Bucharest, Romania | Telecommunications

Current Asentinel

Previous Asentinel

Education Faculty of Mathematics and Computer Science, University of Bucharest

Send a messa

https://ro.linkedin.com/in/sorin-slavic-64132023

Education

Faculty of Mathematics and Computer Science, University of Bucharest
Master's Degree, Database and Web Technologies
2010 – 2012

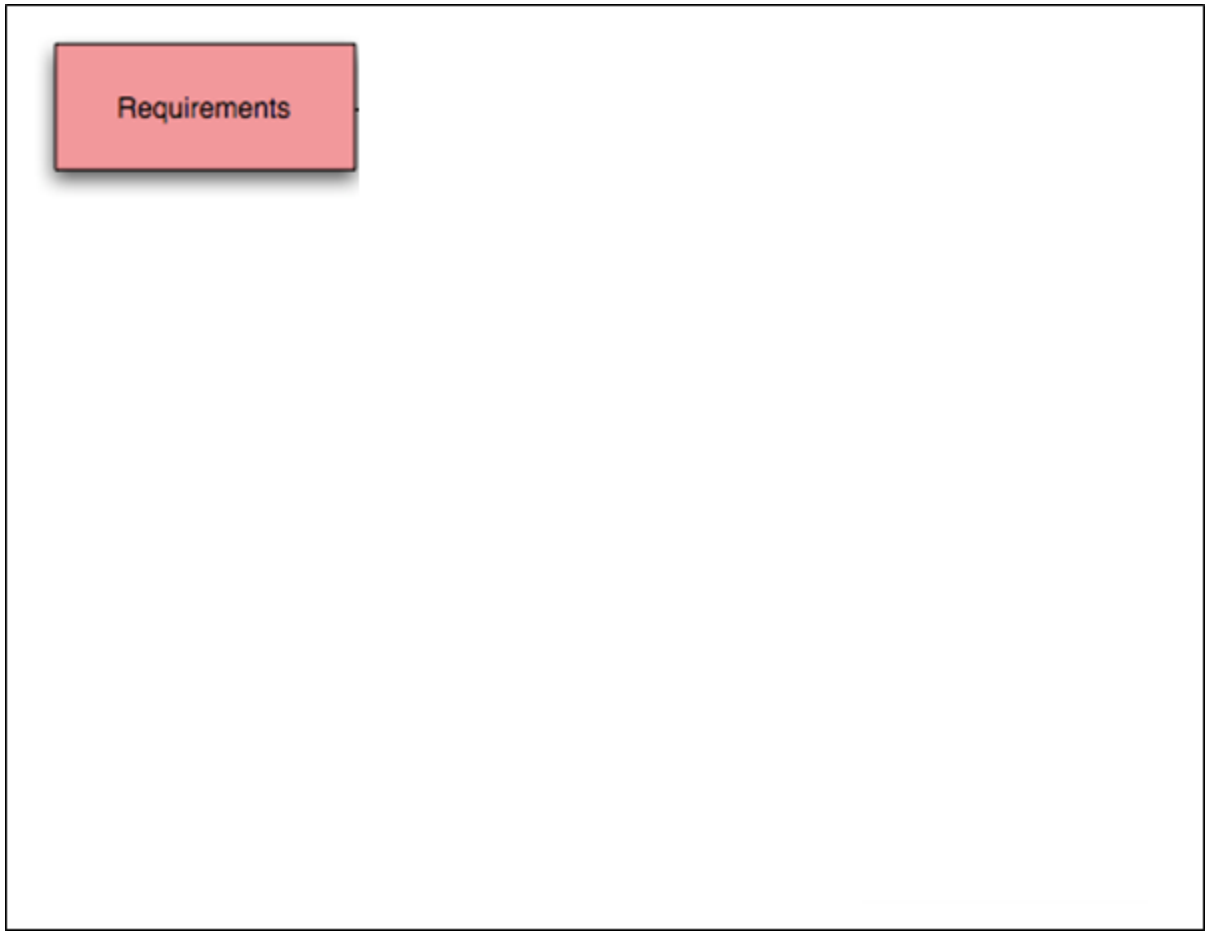
Faculty of Mathematics and Computer Science, University of Bucharest
Bachelor's degree, Computer Science
2007 – 2010

https://www.linkedin.com/in/sorin-slavic-64132023

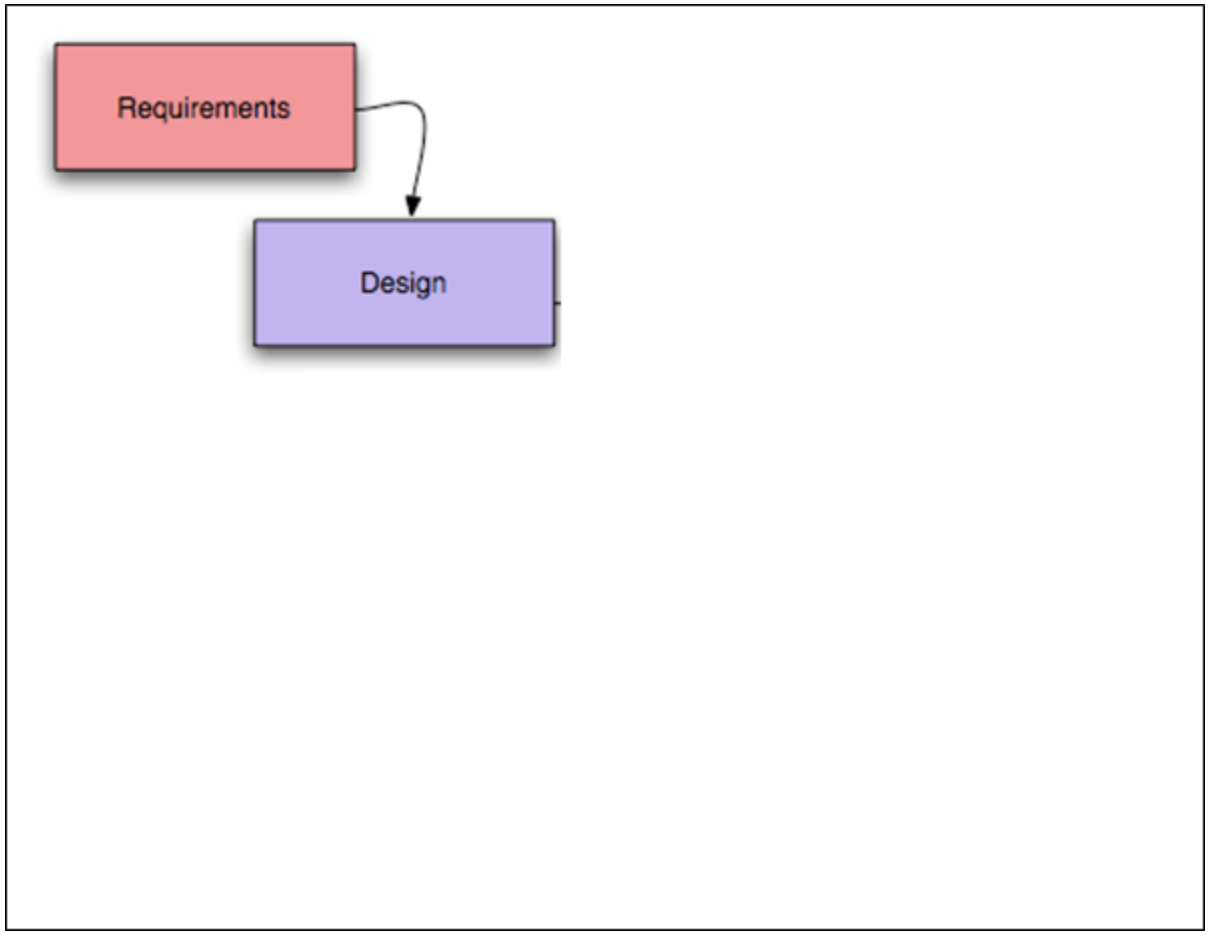
literally

When you say: “this will do that when”
The answer is: I DON’T BELIEVE YOU

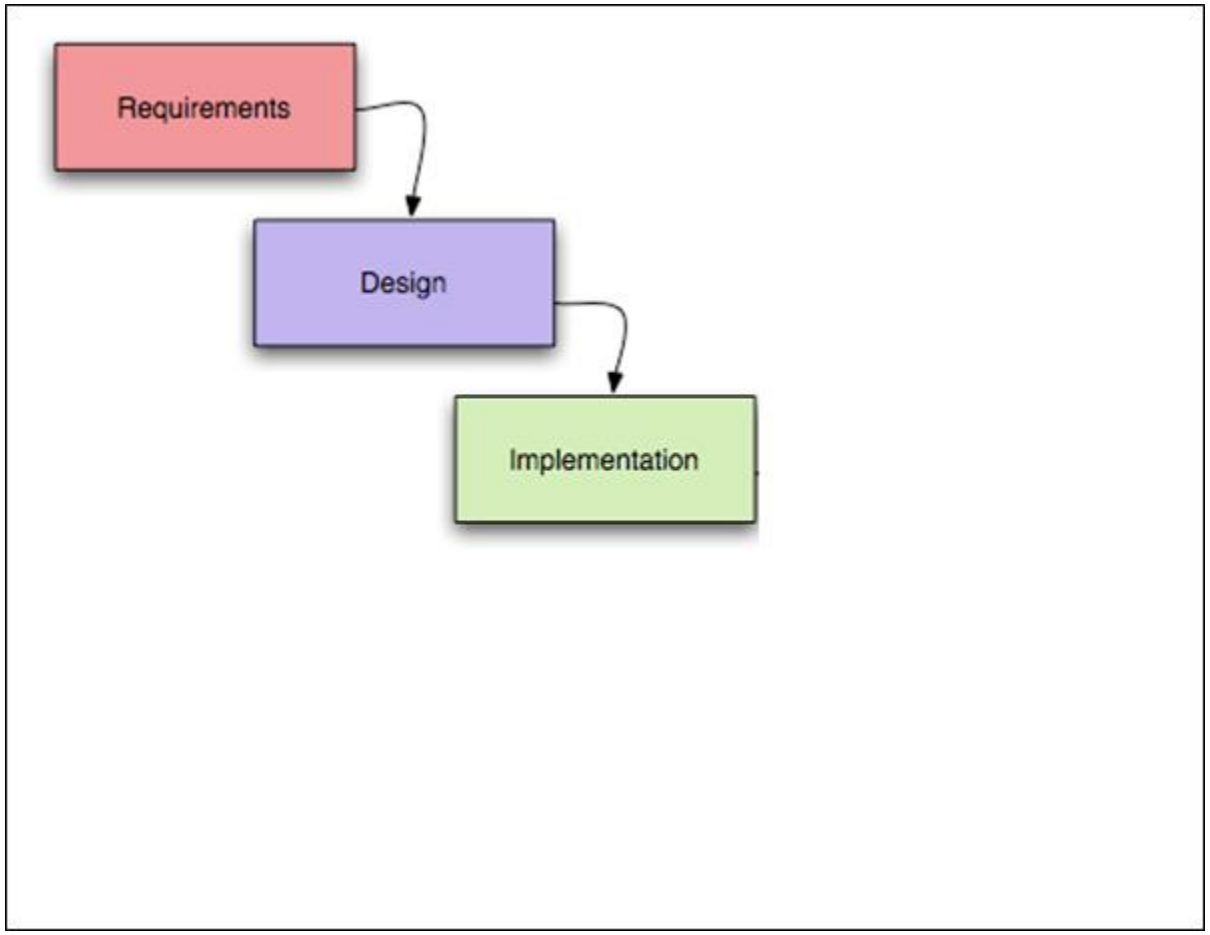
Area sum of all 4 sided figures constructed with 10 points on the XOY plan



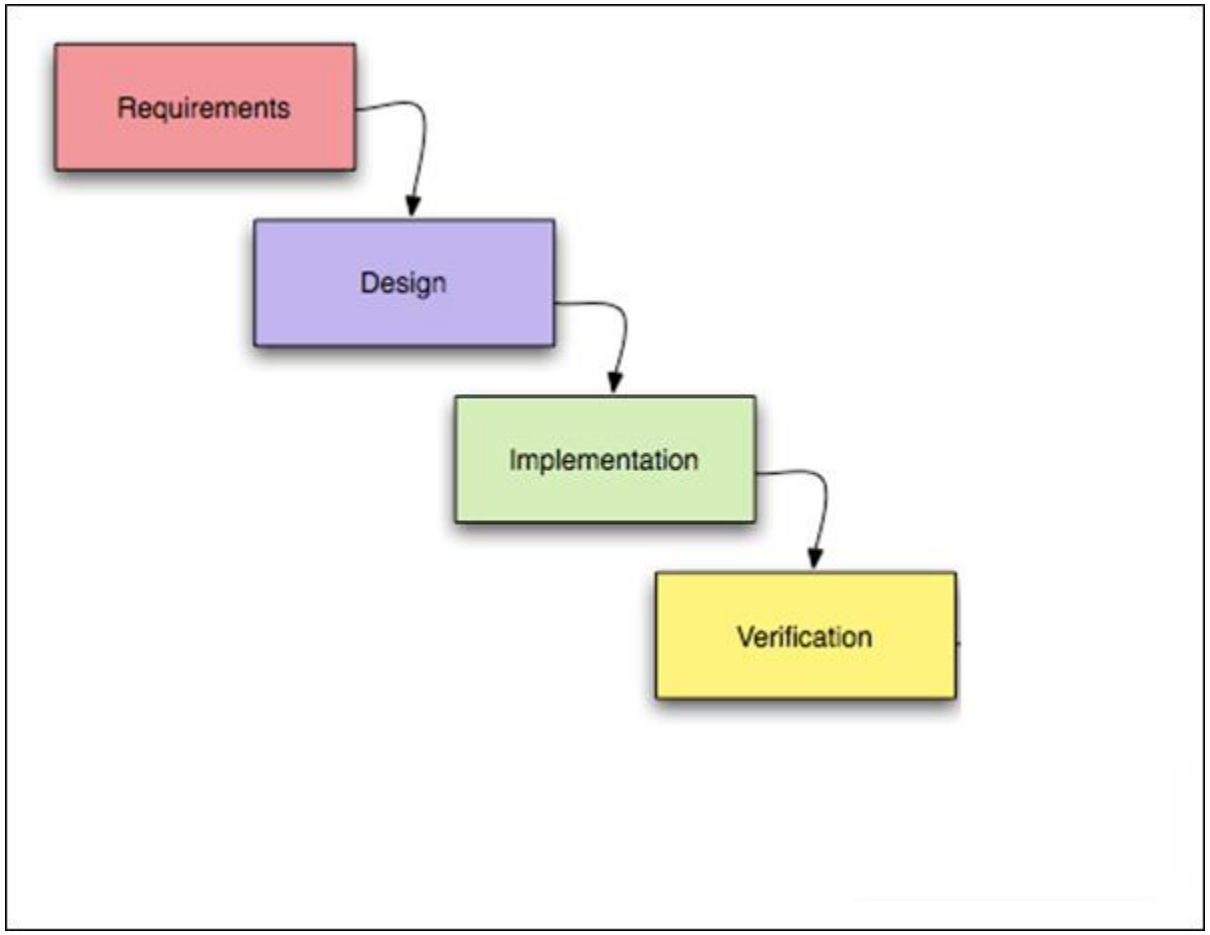
Area sum of all 4 sided figures constructed with 10 points on the XOY plan



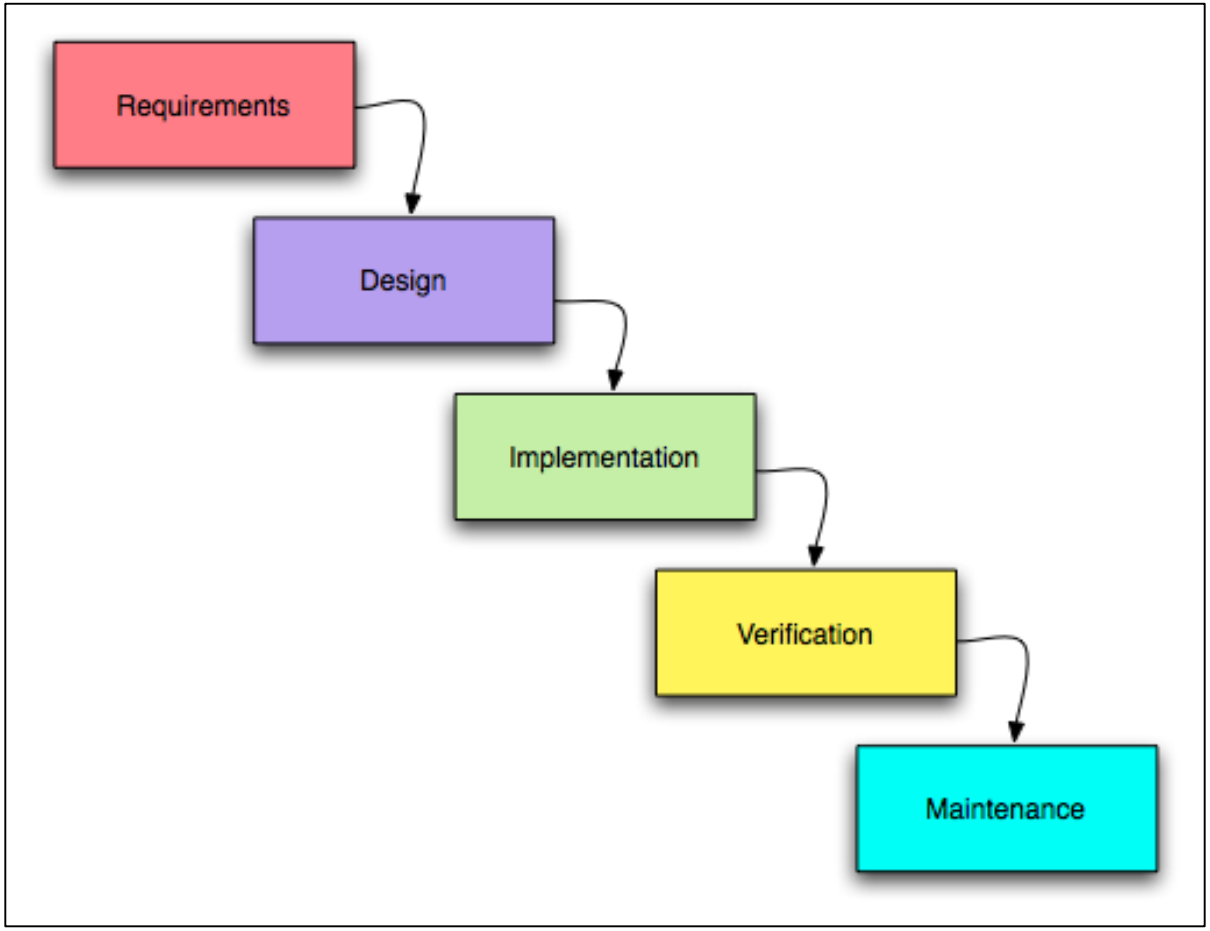
Area sum of all 4 sided figures constructed with 10 points on the XOY plan



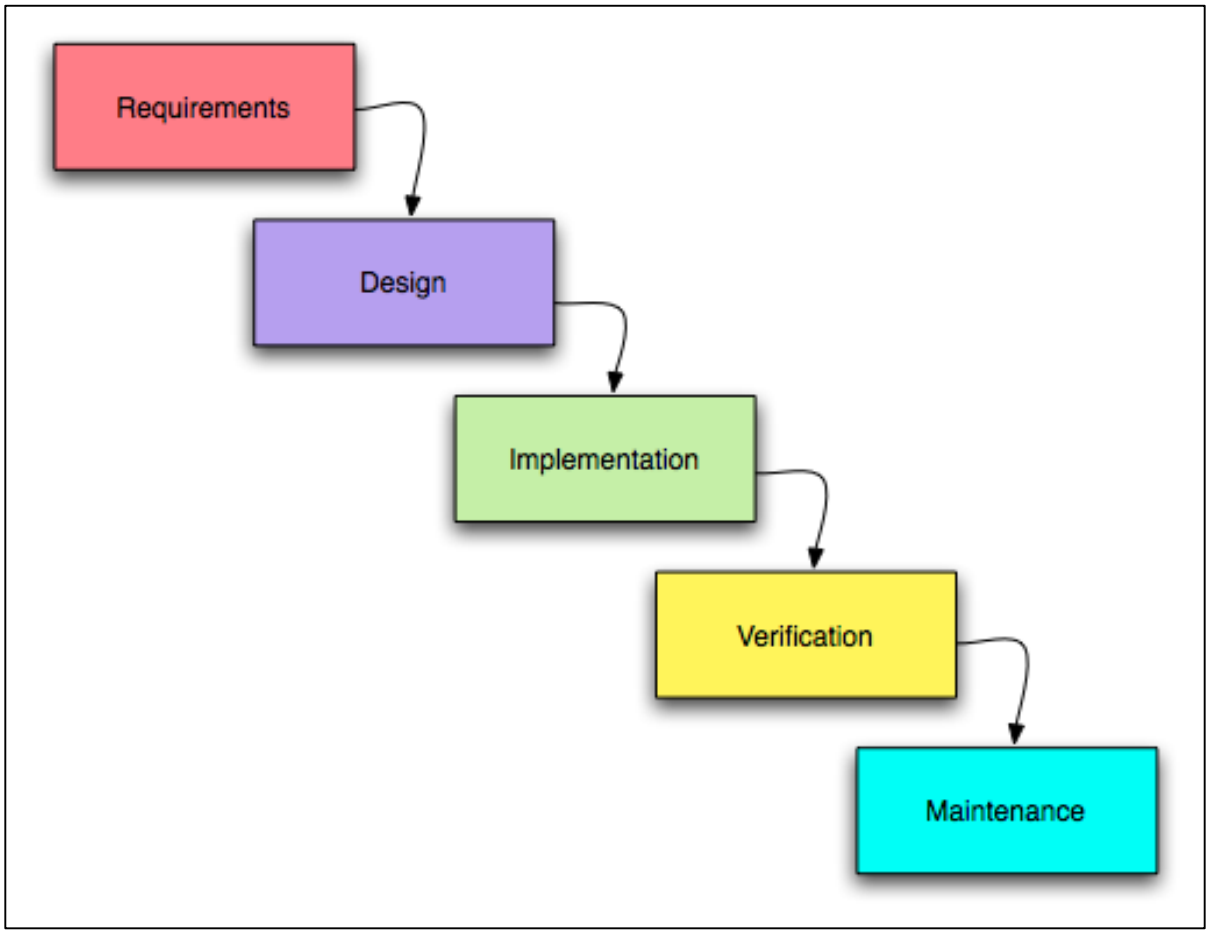
Area sum of all 4 sided figures constructed with 10 points on the XOY plan



Area sum of all 4 sided figures constructed with 10 points on the XOY plan



Waterfall



Design -> Implementation -> Testing

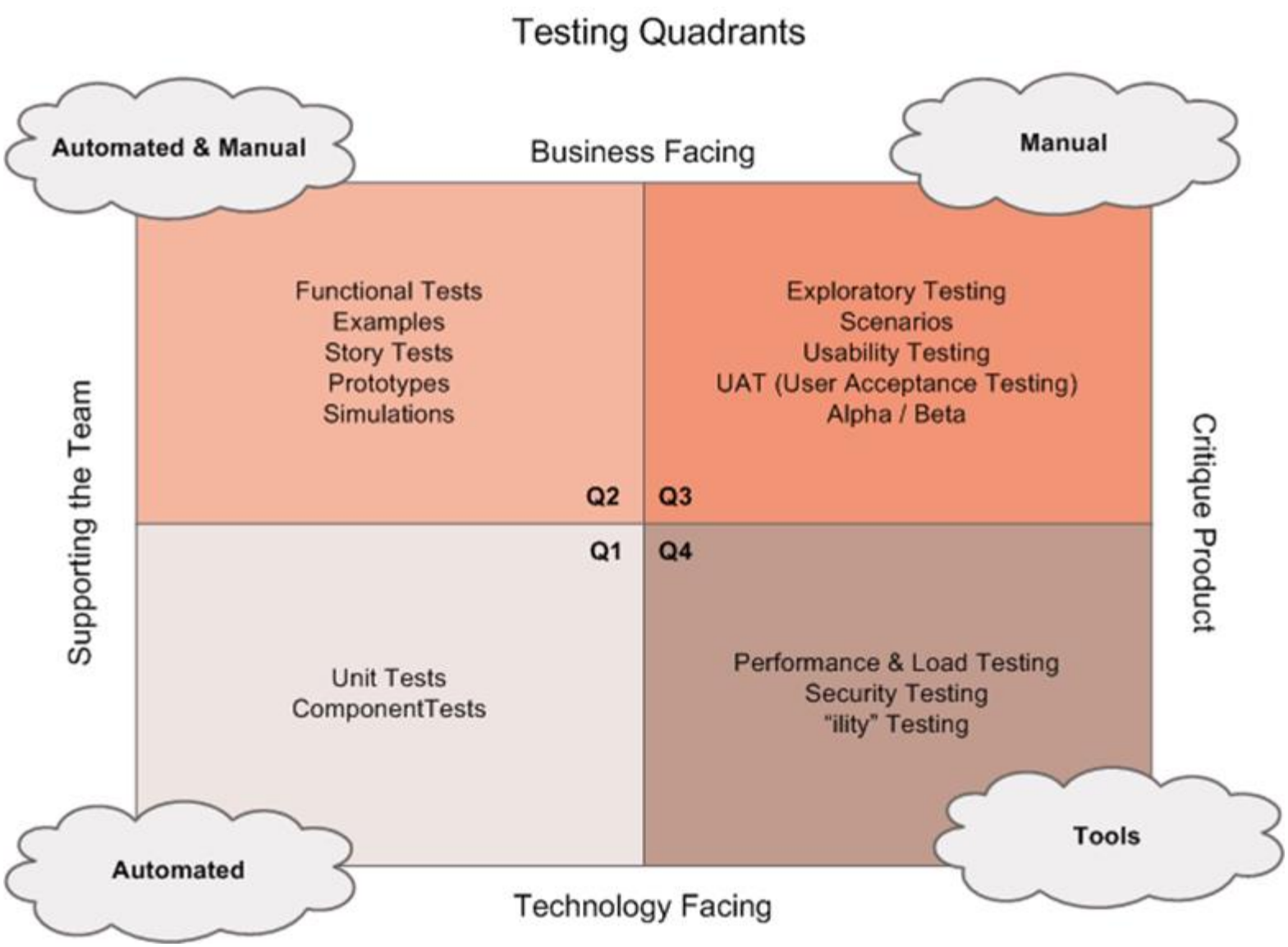
Design Driven Testing

solution-probleming

you have a solution – you try to find problems with it, try to validate it does what it is supposed to do, but also you try to brake it

QA Team

“find” “problems”



Software testing involves the execution of a software component or system component to evaluate one or more properties of interest. In general, these properties indicate the extent to which the component or system under test:

- meets the requirements that guided its design and development,
- responds correctly to all kinds of inputs,
- performs its functions within an acceptable time,
- is sufficiently usable,
- can be installed and run in its intended environments, and
- achieves the general result its stakeholders desire.

Unit testing [edit]

Main article: Unit testing

Unit testing, also known as component testing, refers to tests that verify the functionality of a specific section of code, usually at the function level. In an object-oriented environment, this is usually at the class level, and the minimal unit tests include the constructors and destructors.[32]

Integration testing [edit]

Main article: Integration testing

Integration testing is any type of software testing that seeks to verify the interfaces between components against a software design. Software components may be integrated in an iterative way or all together ("big bang"). Normally the former is considered a better practice since it allows interface issues to be located more quickly and fixed.

https://en.wikipedia.org/wiki/Software_testing

Design -> Implementation -> Testing

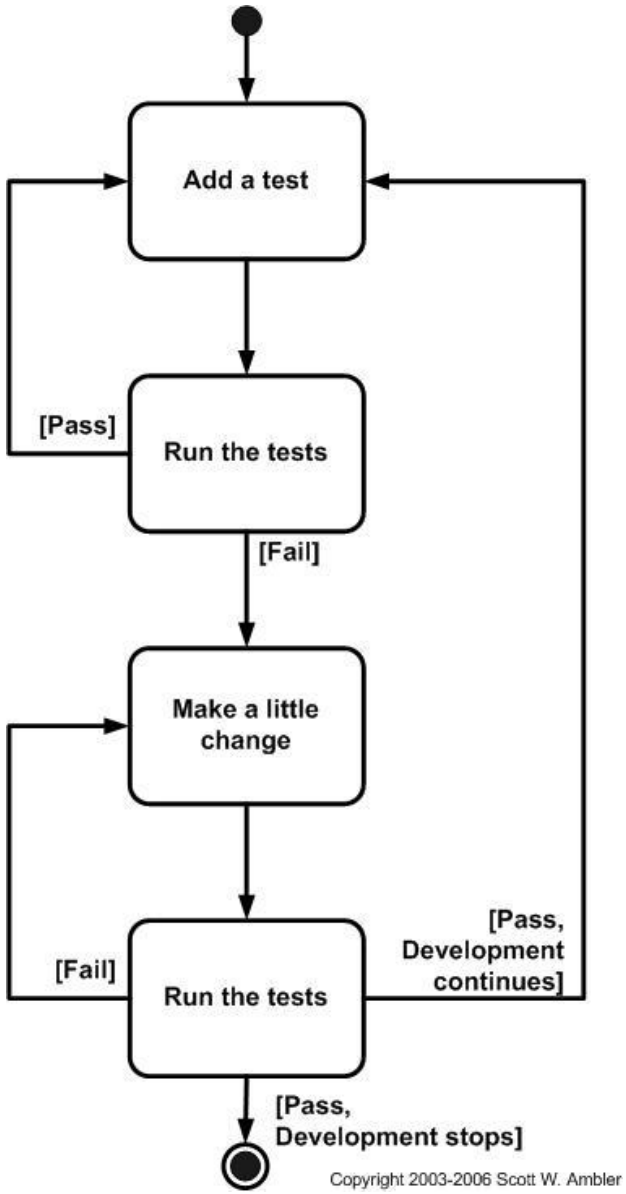
Design -> Implementation -> Testing

Why not ..

Examples -> Testing -> Implementation

“real” “problem solving”

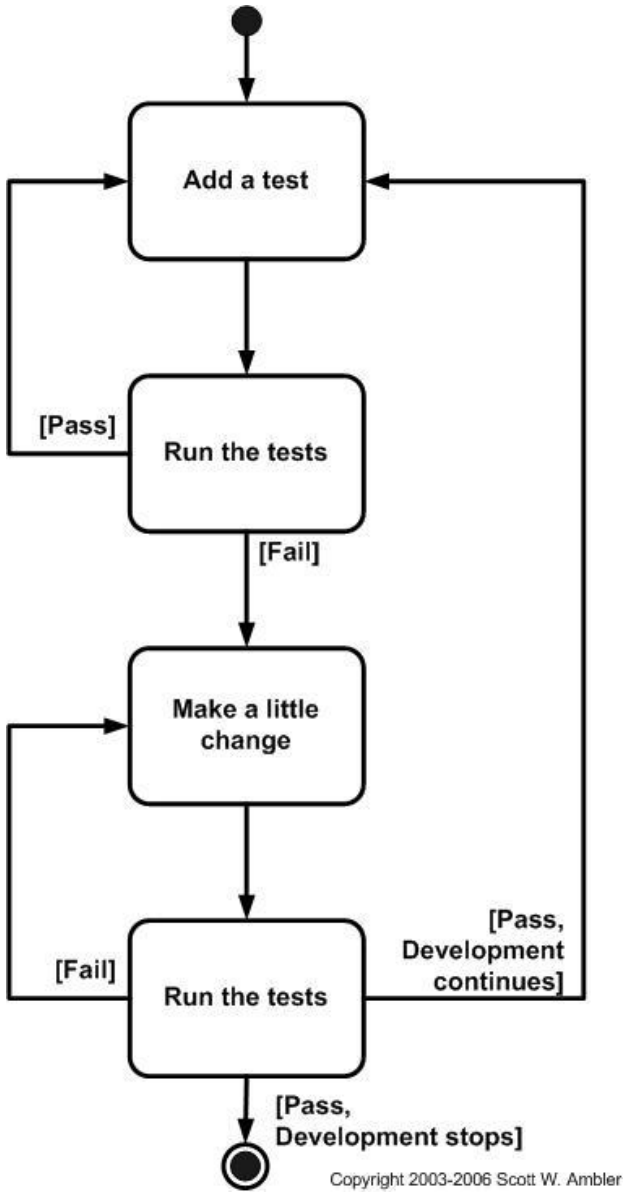
What is TDD?



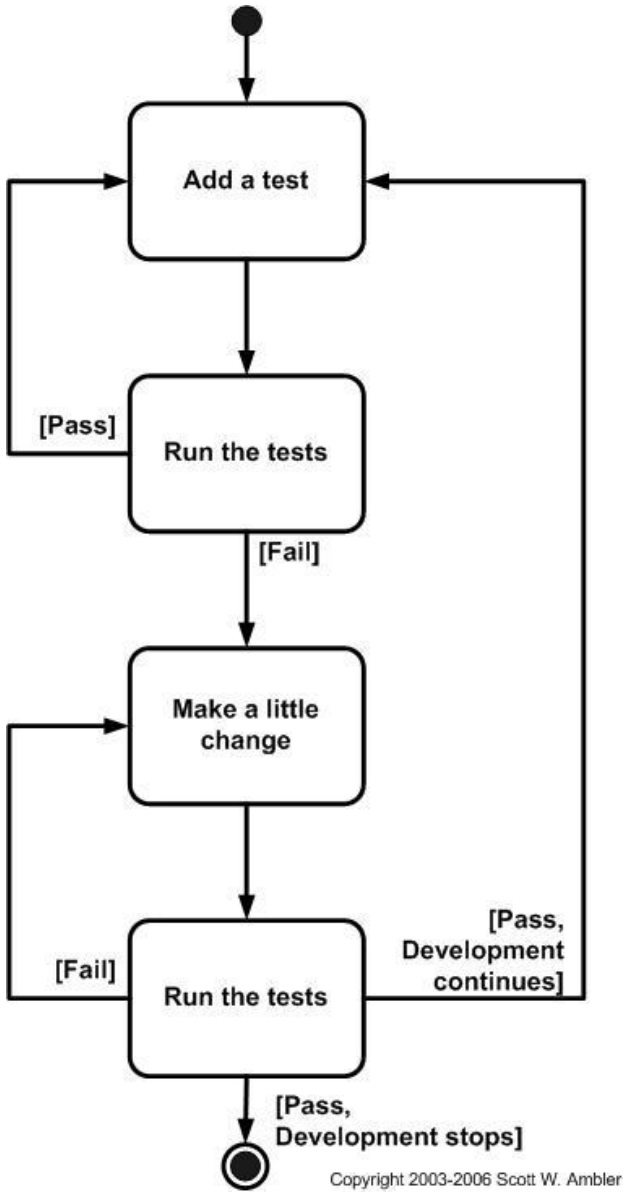
TDD is a mind-hack

TDD is personal

TDD is not a “magic keyboard”

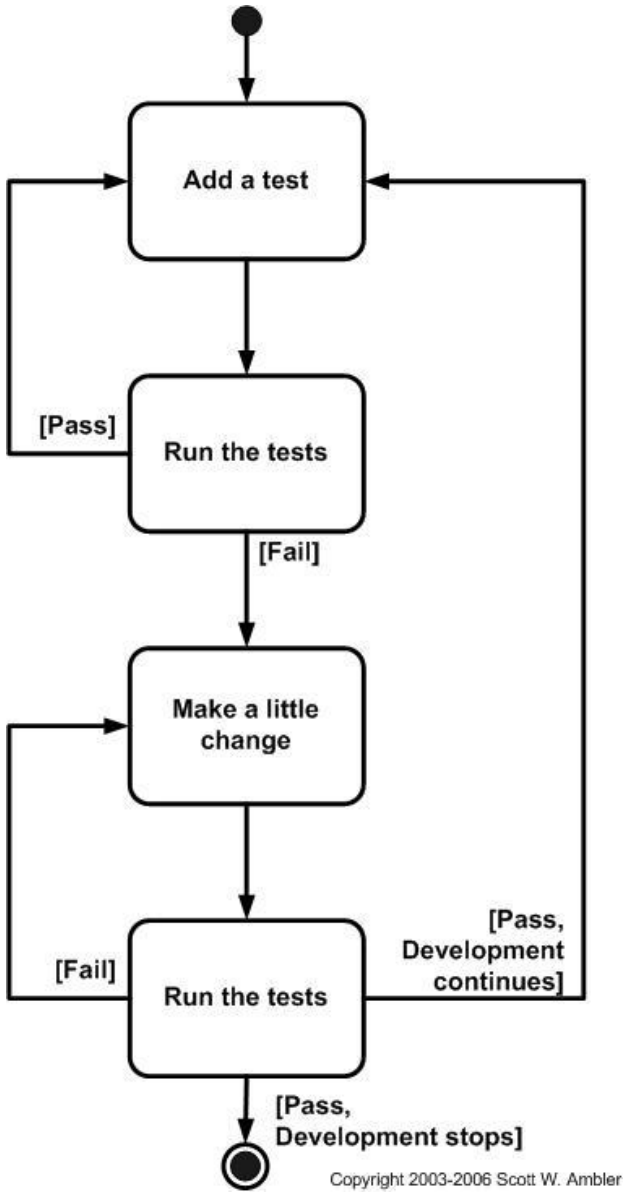


1. You are not allowed to write any production code unless it is to make a failing unit test pass.



1. You are not allowed to write any production code unless it is to make a failing unit test pass.

2. You are not allowed to write any more of a unit test than is sufficient to fail; and compilation failures are failures.

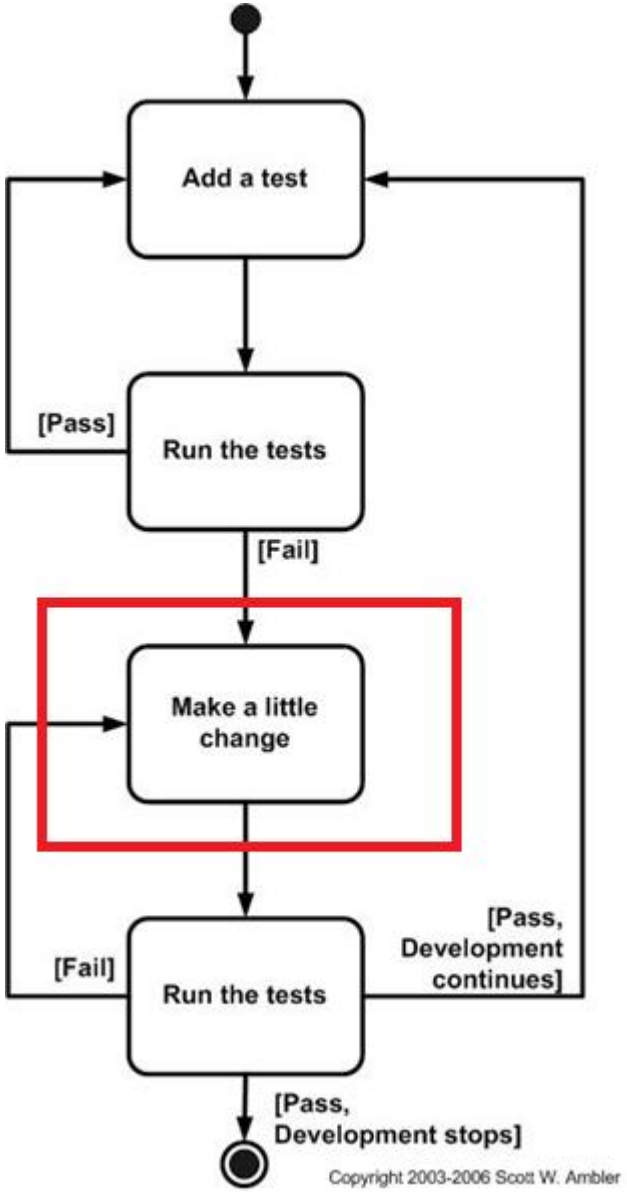


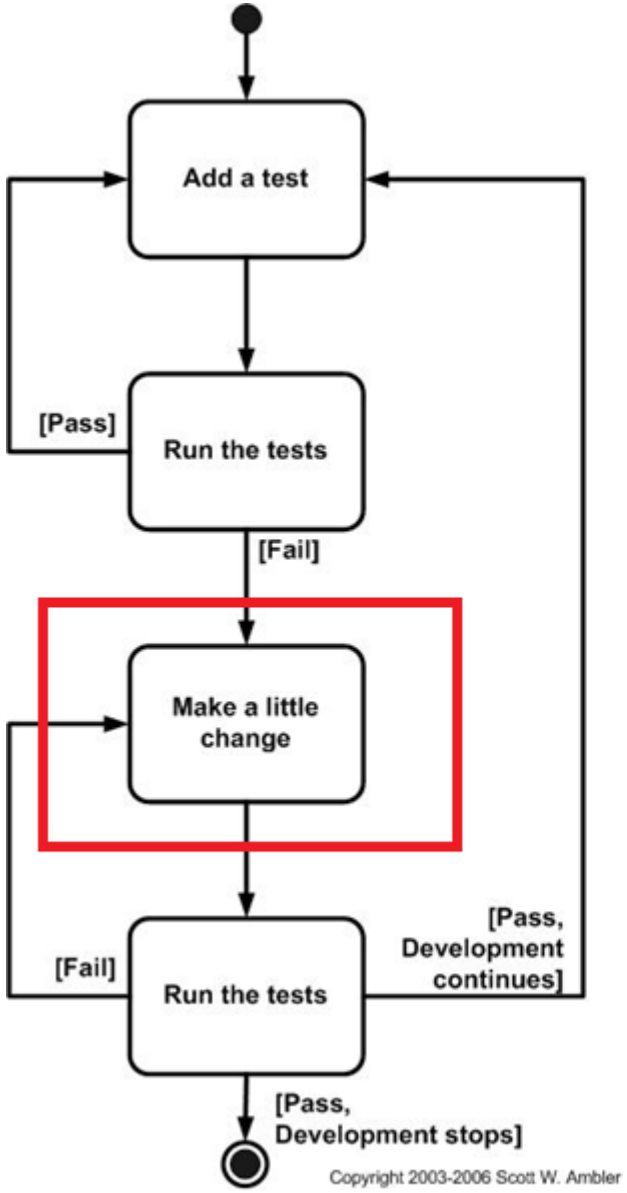
1. You are not allowed to write any production code unless it is to make a failing unit test pass.

2. You are not allowed to write any more of a unit test than is sufficient to fail; and compilation failures are failures.

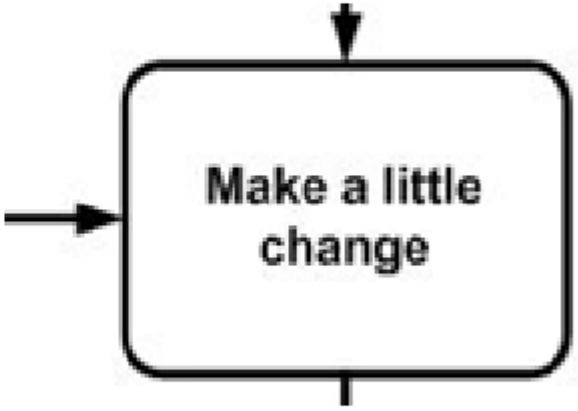
3. You are not allowed to write any more *production code** than is sufficient to pass the one failing unit test.

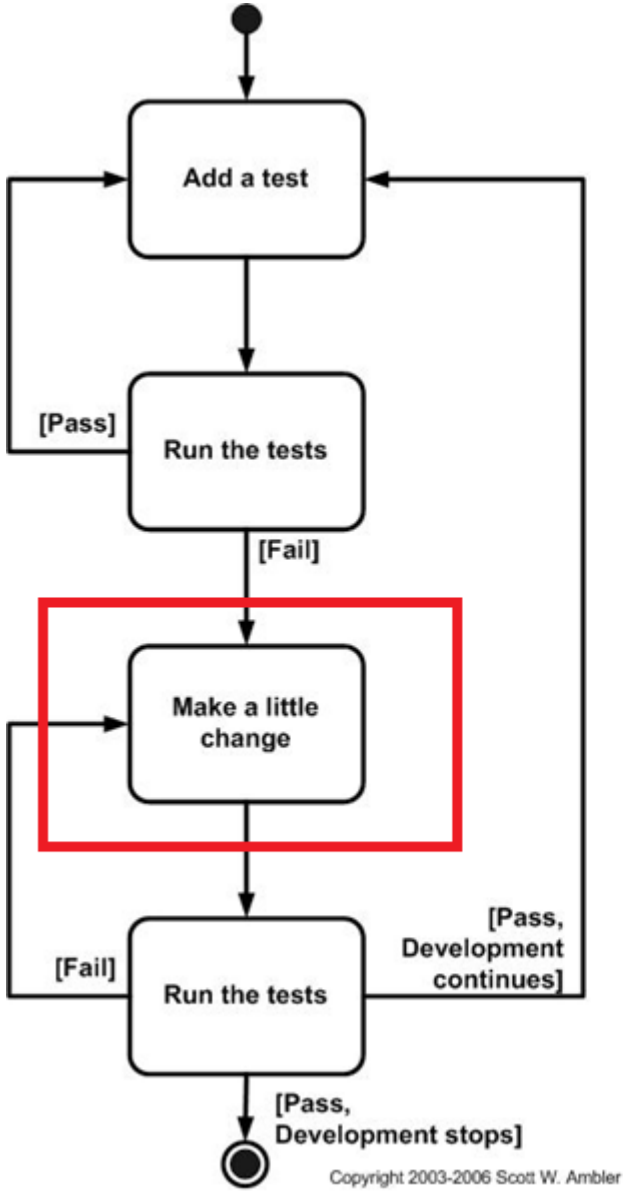
Robert C. Martin



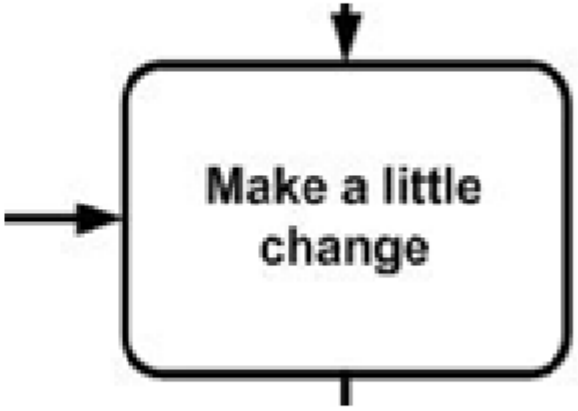


1. Only make a change when a test is failing





1. Only make a change when a test is failing



2. Make the “littlest” change you can. The “simplest” solution to solve the test

But …. If I start with “the simplest solution” – what if it isn’t the best / efficient / correct one ? I should spend the energy and try to figure it out …

> *"We should forget about small efficiencies, say about 97% of the time; premature optimization is the root of all evil" – Donald E. Knuth, Structured Programming with go to Statements*

Rules Of Optimization

The "rules" of optimising are a rhetorical device intended to dissuade novice programmers from cluttering up their programs with vain attempts at writing optimal code. They are:

1. FirstRuleOfOptimization - **Don't.**
2. SecondRuleOfOptimization - **Don't... *yet.***
3. ProfileBeforeOptimizing

> *"We should forget about small efficiencies, say about 97% of the time; premature optimization is the root of all evil" – Donald E. Knuth, Structured Programming with go to Statements*

Rules Of Optimization

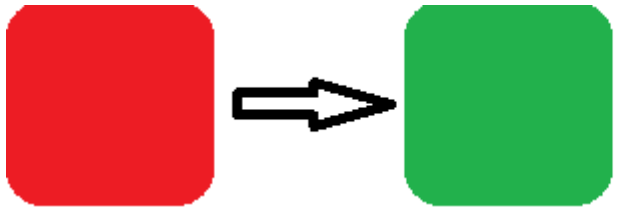
The "rules" of optimising are a rhetorical device intended to dissuade novice programmers from cluttering up their programs with vain attempts at writing optimal code. They are:

1. FirstRuleOfOptimization - **Don't.**
2. SecondRuleOfOptimization - **Don't... *yet.***
3. ProfileBeforeOptimizing

How can you correct / optimize / redesign something that does not work ?

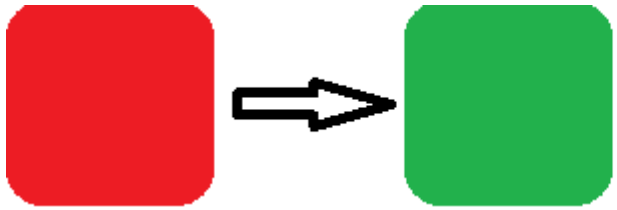
> *"We should forget about small efficiencies, say about 97% of the time; premature optimization is the root of all evil" – Donald E. Knuth, Structured Programming with go to Statements*

Write a
failing
test



Write a failing test

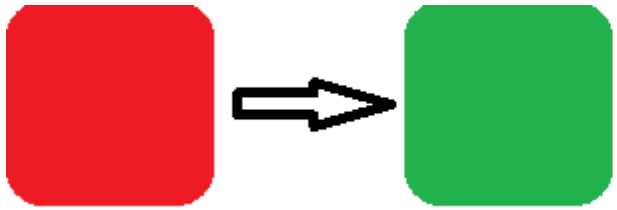
Make the test pass

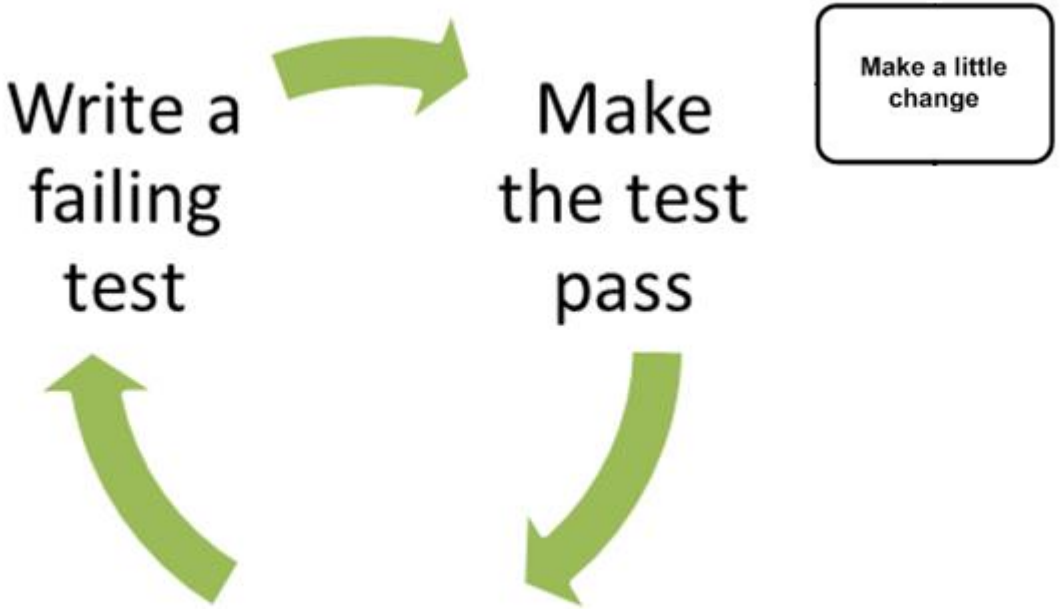


Write a failing test

Make the test pass

Make a little change





How can you correct / optimize / redesign something that does not work ?

How can you correct / optimize / redesign something that does not work ?

…correct something that does not work ….

How can you correct / optimize / redesign something that does not work ?

…correct something that does not work ….

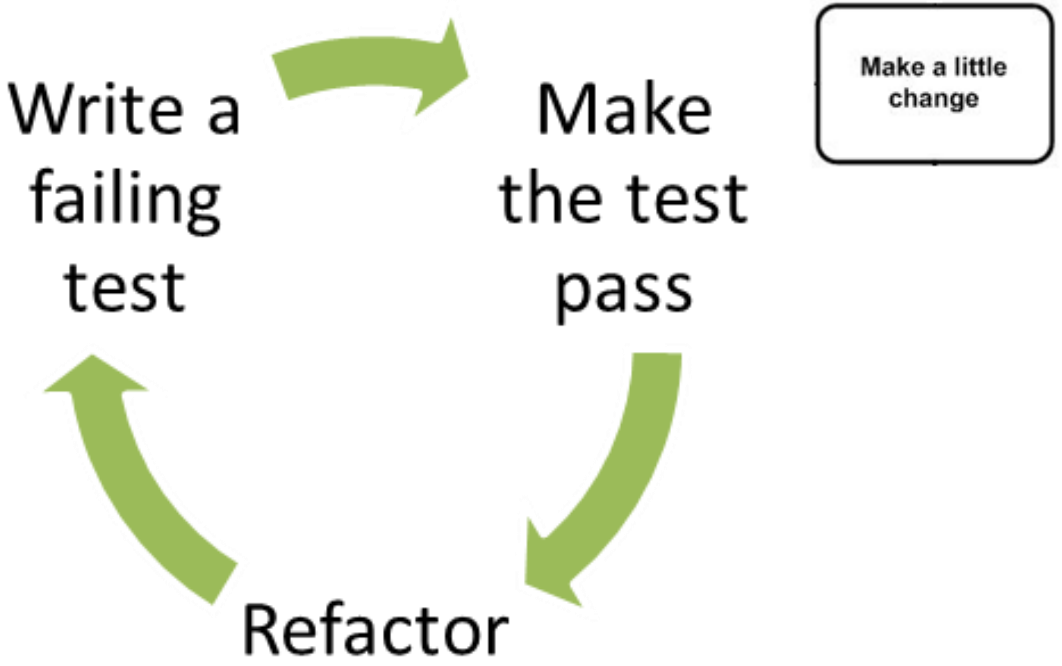
…does not work ….

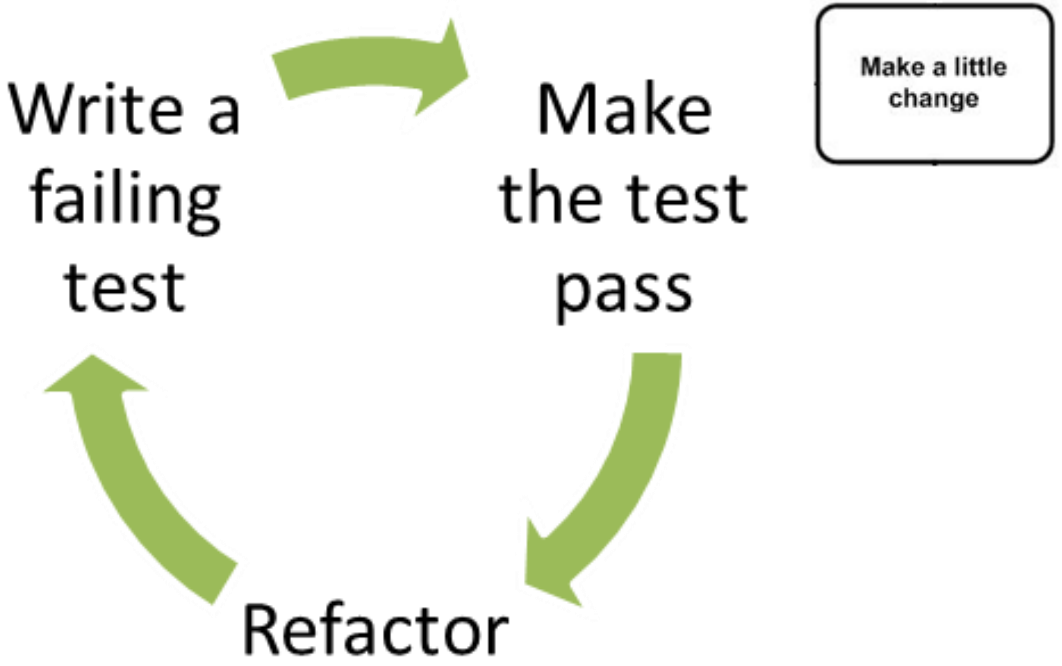
How can you correct / optimize / redesign something that does not work ?

…correct something that does not work ….

…does not work ….

All tests pass ! It does work !





“Everything“ works !
You can refactor – make it the best code you can !
You are not adding new “features” - *production code**

Code Refactoring

Clean Code

K.I.S.S. – keep it simple, stupid ☺

Any fool can write code that a computer can understand. Good programmers write code that humans can understand...

K.I.S.S. – keep it simple, stupid ☺

Any fool can write code that a computer can understand. Good programmers write code that humans can understand...

Boy Scout Rule

Always leave the campground cleaner than you found it..

K.I.S.S. – keep it simple, stupid ☺

Any fool can write code that a computer can understand. Good programmers write code that humans can understand...

Boy Scout Rule

Always leave the campground cleaner than you found it..

High cohesion + Low coupling

K.I.S.S. – keep it simple, stupid ☺

Any fool can write code that a computer can understand. Good programmers write code that humans can understand...

Boy Scout Rule

Always leave the campground cleaner than you found it..

High cohesion + Low coupling

D R Y Don’t repeat yourself

K.I.S.S. – keep it simple, stupid ☺

Any fool can write code that a computer can understand. Good programmers write code that humans can understand...

Boy Scout Rule

Always leave the campground cleaner than you found it..

High cohesion + Low coupling

D R Y Don't repeat yourself

S.O.L.I.D.

Java + JUnit

Prime Factors

Prime Factors

Fundamental theorem of arithmetic

From Wikipedia, the free encyclopedia

In number theory, the **fundamental theorem of arithmetic**, also called the **unique factorization theorem** or the **unique-prime-factorization theorem**, states that every integer greater than 1[note 1] either is prime itself or is the product of prime numbers, and that this product is unique, up to the order of the factors.[3][4][5] For example,

$1200 = 2^4 \times 3^1 \times 5^2 = 3 \times 2 \times 2 \times 2 \times 2 \times 5 \times 5 = 5 \times 2 \times 3 \times 2 \times 5 \times 2 \times 2 = \text{etc.}$

1 – has no prime factors
2 – just 2
3 – just 3
4 – 2, 2
5 – just 5
6 – 2, 3
8 – 2, 2, 2

1540	**2**
770	**2**
385	**5**
77	**7**
11	**11**
1	

Prime Factors

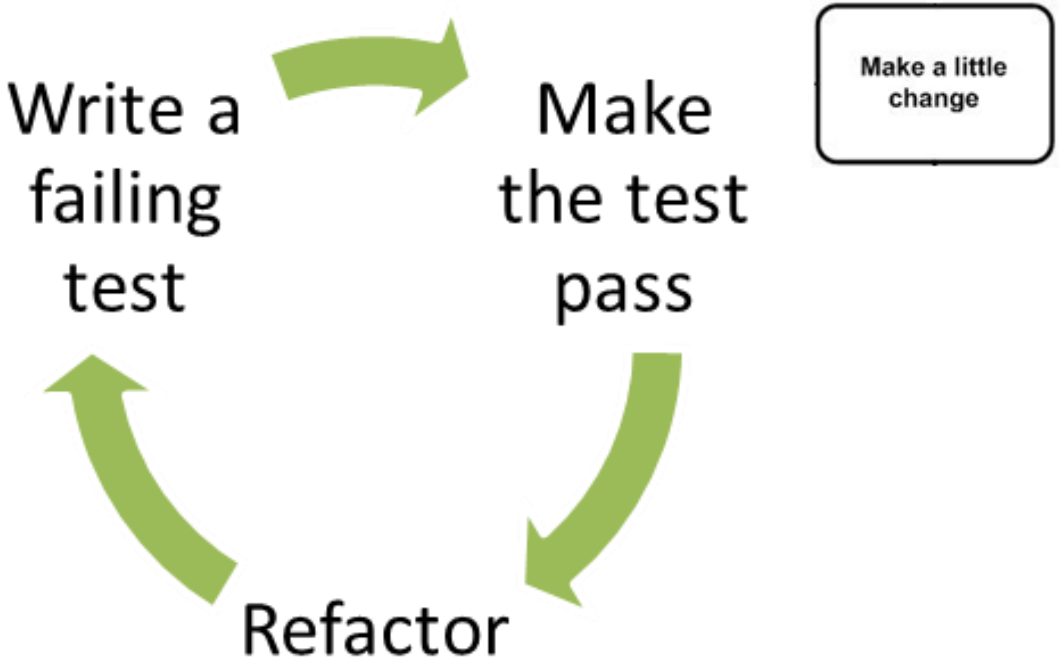
Given a positive integer number, return a list of all of the prime factors of that number. The returned prime values will be ordered ascending.

Ex:
input = 1540
Returned list: 2, 2, 5, 7, 11

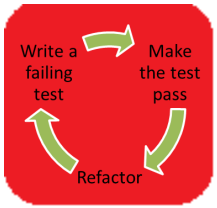
Prime Factors

Solution – obviously not optimal:

```
read n
for k = 2 .. n
    if (k is prime)
        copyn = n
        while (copyn % k = 0)
            write k
            copyn = copyn / k
        endwhile
    endif
endfor
end
```



Prime Factors



PrimeFactorsTestCase.java

```
1  package tdd.primefactors;
2
3  import static org.junit.Assert.assertEquals;
4
5  import java.util.Arrays;
6
7  import org.junit.Test;
8  // add the first test - compilation failure
9  public class PrimeFactorsTestCase {
10
11     @Test
12     public void testOneHasNoPrimeFactors() {
13         assertEquals(Arrays.asList(),
14                 PrimeFactors.of(1));
15     }
16 }
17
```

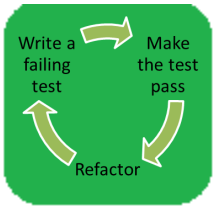
PrimeFactors cannot be resolved

11 quick fixes available:

- Create class 'PrimeFactors'
- Create interface 'PrimeFactors'
- Create constant 'PrimeFactors'
- Create enum 'PrimeFactors'
- Create local variable 'PrimeFactors'
- Change to 'PrimeFactorss' (tdd.primefactors)
- Change to 'PrimeFactorsTestCase' (tdd.primefactors)
- Create field 'PrimeFactors'

Press 'F2' for focus

Prime Factors



PrimeFactorsTestCase.java

```
1  package tdd.primefactors;
2
3  import static org.junit.Assert.assertEquals;
4
5  import java.util.Arrays;
6
7  import org.junit.Test;
8
9  public class PrimeFactorsTestCase {
10
11     @Test
12     public void testOneHasNoPrimeFactors() {
13         assertEquals(Arrays.asList(),
14                 PrimeFactors.of(1));
15     }
16 }
17
```

PrimeFactors.java

```
1  package tdd.primefactors;
2
3  import java.util.ArrayList;
5
6  // implement the basic method - return empty list
7  public class PrimeFactors {
8      public static List<Integer> of(int n) {
9          return new ArrayList<Integer>();
10     }
11 }
12
```

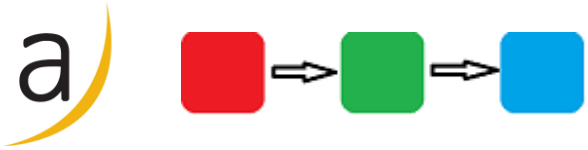
JUnit

Finished after 0.028 seconds

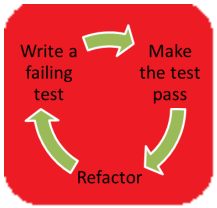
Runs: 1/1 Errors: 0 Failures: 0

tdd.primefactors.PrimeFactorsTestCase [Runner: JUnit 4] (0.000 s)

Failure Trace



Prime Factors



PrimeFactorsTestCase.java

```
1  package tdd.primefactors;
2
3  import static org.junit.Assert.assertEquals;
8  // add the second test - validate that: 2 -> 2
9  public class PrimeFactorsTestCase {
10
11     @Test
12     public void testOneHasNoPrimeFactors() {
13         assertEquals(Arrays.asList(),
14                 PrimeFactors.of(1));
15     }
16
17     @Test
18     public void testTwoHasJustOneFactorItself() {
19         assertEquals(Arrays.asList(2),
20                 PrimeFactors.of(2));
21     }
22 }
23
```

PrimeFactors.java

```
1  package tdd.primefactors;
2
3  import java.util.ArrayList;
5
6  public class PrimeFactors {
7      public static List<Integer> of(int n) {
8          return new ArrayList<Integer>();
9      }
10 }
11
```

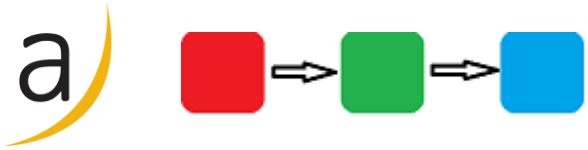
JUnit

Finished after 0.031 seconds

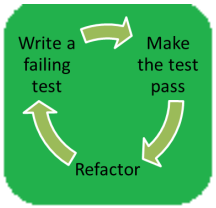
Runs: 2/2 Errors: 0 Failures: 1

tdd.primefactors.PrimeFactorsTestCase [Runner: JUnit 4] (0.006 s)
testTwoHasJustOneFactorItself (0.005 s)
testOneHasNoPrimeFactors (0.000 s)

Failure Trace
java.lang.AssertionError: expected:<[2]> but was:<[]>
at tdd.primefactors.PrimeFactorsTestCase.testTwoHasJustOneFactorItself(Pr



Prime Factors



PrimeFactorsTestCase.java

```
1  package tdd.primefactors;
2
3  import static org.junit.Assert.assertEquals;
8
9  public class PrimeFactorsTestCase {
10
11     @Test
12     public void testOneHasNoPrimeFactors() {
13         assertEquals(Arrays.asList(),
14                 PrimeFactors.of(1));
15     }
16
17     @Test
18     public void testTwoHasJustOneFactorItself() {
19         assertEquals(Arrays.asList(2),
20                 PrimeFactors.of(2));
21     }
22 }
23
```

PrimeFactors.java

```
1  package tdd.primefactors;
2
3  import java.util.ArrayList;
5  // fix the second test - for input "2"
6  public class PrimeFactors {
7
8      public static List<Integer> of(int n) {
9          List<Integer> primeFactors = new ArrayList<Integer>();
10         if (n > 1)
11             primeFactors.add(2);
12
13         return primeFactors;
14     }
```

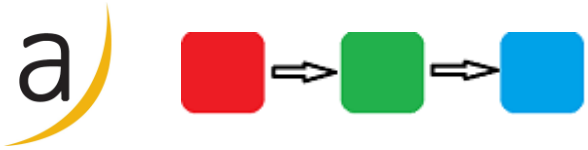
JUnit

Finished after 0.03 seconds

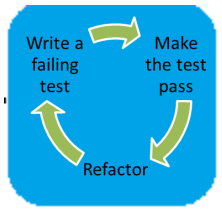
Runs: 2/2 Errors: 0 Failures: 0

tdd.primefactors.PrimeFactorsTestCase [Runner: JUnit 4] (0.000 s)

Failure Trace



Prime Factors



PrimeFactorsTestCase.java

```
1  package tdd.primefactors;
2
3  import static java.util.Arrays.asList;
4  import static org.junit.Assert.assertEquals;
5  import static tdd.primefactors.PrimeFactors.of;
6
7  import org.junit.Test;
8  // all the tests are passing - we can refactor a bit
9  public class PrimeFactorsTestCase {
10
11     @Test
12     public void testOneHasNoPrimeFactors() {
13         assertEquals(asList(), of(1));
14     }
15
16     @Test
17     public void testTwoHasJustOneFactorItself() {
18         assertEquals(asList(2), of(2));
19     }
20 }
21
```

PrimeFactors.java

```
1  package tdd.primefactors;
2
3  import java.util.ArrayList;
5
6  public class PrimeFactors {
7
8      public static List<Integer> of(int n) {
9          List<Integer> primeFactors = new ArrayList<Integer>();
10         if (n > 1)
11             primeFactors.add(2);
12
13         return primeFactors;
14     }
```

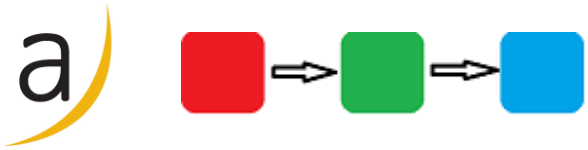
JUnit

Finished after 0.03 seconds

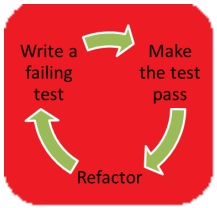
Runs: 2/2 Errors: 0 Failures: 0

tdd.primefactors.PrimeFactorsTestCase [Runner: JUnit 4] (0.000 s)

Failure Trace



Prime Factors



PrimeFactorsTestCase.java

```
package tdd.primefactors;

import static java.util.Arrays.asList;
import static org.junit.Assert.assertEquals;
import static tdd.primefactors.PrimeFactors.of;

import org.junit.Test;
// the next test - 3 -> 3
public class PrimeFactorsTestCase {

    @Test
    public void testOneHasNoPrimeFactors() {
        assertEquals(asList(), of(1));
    }

    @Test
    public void testTwoHasJustOneFactorItself() {
        assertEquals(asList(2), of(2));
    }

    @Test
    public void testThreeHasJustOneFactorItself() {
        assertEquals(asList(3), of(3));
    }
```

PrimeFactors.java

```
package tdd.primefactors;

import java.util.ArrayList;

public class PrimeFactors {

    public static List<Integer> of(int n) {
        List<Integer> primeFactors = new ArrayList<Integer>();
        if (n > 1)
            primeFactors.add(2);

        return primeFactors;
    }
```

JUnit

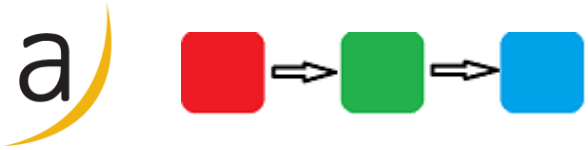
Finished after 0.032 seconds

Runs: 3/3 Errors: 0 Failures: 1

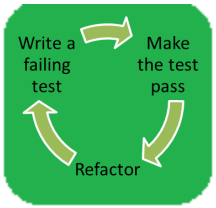
tdd.primefactors.PrimeFactorsTestCase [Runner: JUnit 4] (0.006 s)
- testTwoHasJustOneFactorItself (0.000 s)
- testOneHasNoPrimeFactors (0.000 s)
- testThreeHasJustOneFactorItself (0.006 s)

Failure Trace

java.lang.AssertionError: expected:<[3]> but was:<[2]>
at tdd.primefactors.PrimeFactorsTestCase.testThreeHasJustOneFactorItself(



Prime Factors



PrimeFactorsTestCase.java

```
1  package tdd.primefactors;
2
3  import static java.util.Arrays.asList;
8
9  public class PrimeFactorsTestCase {
10
11     @Test
12     public void testOneHasNoPrimeFactors() {
13         assertEquals(asList(), of(1));
14     }
15
16     @Test
17     public void testTwoHasJustOneFactorItself() {
18         assertEquals(asList(2), of(2));
19     }
20
21     @Test
22     public void testThreeHasJustOneFactorItself() {
23         assertEquals(asList(3), of(3));
24     }
25 }
26
```

PrimeFactors.java

```
2
3  import java.util.ArrayList;
5  // correct for input 3
6  public class PrimeFactors {
7
8      public static List<Integer> of(int n) {
9          List<Integer> primeFactors = new ArrayList<Integer>();
10         if (n > 1)
11             primeFactors.add(n);
12
13         return primeFactors;
14     }
15 }
```

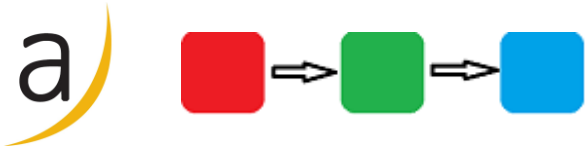
JUnit

Finished after 0.028 seconds

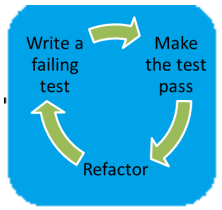
Runs: 3/3 Errors: 0 Failures: 0

tdd.primefactors.PrimeFactorsTestCase [Runner: JUnit 4] (0.001 s)

Failure Trace



Prime Factors



PrimeFactorsTestCase.java

```
1  package tdd.primefactors;
2
3  import static java.util.Arrays.asList;
8  // not indicated - i just did it to save space
9  public class PrimeFactorsTestCase {
10
11     @Test
12     public void testPrimeFactorsGenerator() {
13         assertEquals("factors of 1",
14                 asList(), of(1));
15         assertEquals("factors of 2",
16                 asList(2), of(2));
17         assertEquals("factors of 3",
18                 asList(3), of(3));
19     }
20 }
21
```

PrimeFactors.java

```
2
3  import java.util.ArrayList;
5
6  public class PrimeFactors {
7
8      public static List<Integer> of(int n) {
9          List<Integer> primeFactors = new ArrayList<Integer>();
10         if (n > 1)
11             primeFactors.add(n);
12
13         return primeFactors;
14     }
15 }
```

JUnit

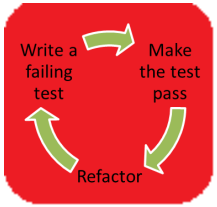
Finished after 0.024 seconds

Runs: 1/1 Errors: 0 Failures: 0

tdd.primefactors.PrimeFactorsTestCase [Runner: JUnit 4] (0.001 s)

Failure Trace

Prime Factors



PrimeFactorsTestCase.java

```
1  package tdd.primefactors;
2
3  import static java.util.Arrays.asList;
8  // the test for 5 would pass, let's try input 4
9  public class PrimeFactorsTestCase {
10
11     @Test
12     public void testPrimeFactorsGenerator() {
13         assertEquals("factors of 1",
14                 asList(), of(1));
15         assertEquals("factors of 2",
16                 asList(2), of(2));
17         assertEquals("factors of 3",
18                 asList(3), of(3));
19         assertEquals("factors of 4",
20                 asList(2, 2), of(4));
21     }
22 }
23
```

PrimeFactors.java

```
2
3  import java.util.ArrayList;
5
6  public class PrimeFactors {
7
8      public static List<Integer> of(int n) {
9          List<Integer> primeFactors = new ArrayList<Integer>();
10         if (n > 1)
11             primeFactors.add(n);
12
13         return primeFactors;
14     }
15 }
```

JUnit

Finished after 0.029 seconds

Runs: 1/1 Errors: 0 Failures: 1

tdd.primefactors.PrimeFactorsTestCase [Runner: JUnit 4] (0.005 s)

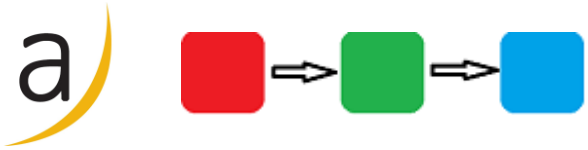
testPrimeFactorsGenerator (0.005 s)

Failure Trace

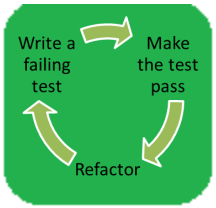
java.lang.AssertionError: factors of 4 expected:<[2, 2]> but was:<[4]>

at tdd.primefactors.PrimeFactorsTestCase.testPrimeFactorsGenerator(Prime

java.lang.AssertionError: factors of 4 expected:<[2, 2]> but was:<[4]>



Prime Factors



PrimeFactorsTestCase.java

```
1  package tdd.primefactors;
2
3  import static java.util.Arrays.asList;
8
9  public class PrimeFactorsTestCase {
10
11     @Test
12     public void testPrimeFactorsGenerator() {
13         assertEquals("factors of 1",
14                 asList(), of(1));
15         assertEquals("factors of 2",
16                 asList(2), of(2));
17         assertEquals("factors of 3",
18                 asList(3), of(3));
19         assertEquals("factors of 4",
20                 asList(2, 2), of(4));
21     }
22 }
23
```

PrimeFactors.java

```
1  package tdd.primefactors;
2
3  import java.util.ArrayList;
5
6  public class PrimeFactors {
7      // fix for input 4 -> 2, 2
8      public static List<Integer> of(int n) {
9          List<Integer> primeFactors = new ArrayList<Integer>();
10         if (n > 1) {
11             if (n % 2 == 0) {
12                 primeFactors.add(2);
13                 n = n/2;
14             }
15             if (n > 1)
16                 primeFactors.add(n);
17         }
18
19         return primeFactors;
20     }
21 }
```

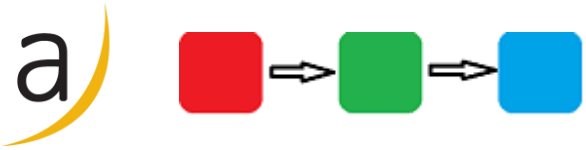
JUnit

Finished after 0.037 seconds

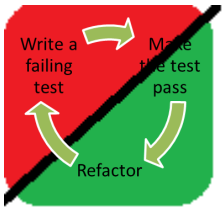
Runs: 1/1 Errors: 0 Failures: 0

tdd.primefactors.PrimeFactorsTestCase [Runner: JUnit 4] (0.003 s)

Failure Trace



Prime Factors



PrimeFactorsTestCase.java

```
1  package tdd.primefactors;
2
3  import static java.util.Arrays.asList;
8
9  public class PrimeFactorsTestCase {
10     // factors of 6 will apparently work
11     @Test
12     public void testPrimeFactorsGenerator() {
13         assertEquals("factors of 1",
14                 asList(), of(1));
15         assertEquals("factors of 2",
16                 asList(2), of(2));
17         assertEquals("factors of 3",
18                 asList(3), of(3));
19         assertEquals("factors of 4",
20                 asList(2, 2), of(4));
21         assertEquals("factors of 6",
22                 asList(2, 3), of(6));
23     }
24 }
25
```

*PrimeFactors.java

```
1  package tdd.primefactors;
2
3  import java.util.ArrayList;
5
6  public class PrimeFactors {
7
8      public static List<Integer> of(int n) {
9          List<Integer> primeFactors = new ArrayList<Integer>();
10         if (n > 1) {
11             if (n % 2 == 0) {
12                 primeFactors.add(2);
13                 n = n/2;
14             }
15             if (n > 1)
16                 primeFactors.add(n);
17         }
18
19         return primeFactors;
20     }
21 }
```

JUnit

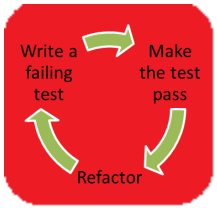
Finished after 0.021 seconds

Runs: 1/1 Errors: 0 Failures: 0

tdd.primefactors.PrimeFactorsTestCase [Runner: JUnit 4] (0.001 s)

Failure Trace

Prime Factors



PrimeFactorsTestCase.java

```
1  package tdd.primefactors;
2
3  import static java.util.Arrays.asList;
8
9  public class PrimeFactorsTestCase {
10     // factors of 8 -> 2, 2, 2
11     @Test
12     public void testPrimeFactorsGenerator() {
13         assertEquals("factors of 1",
14                 asList(), of(1));
15         assertEquals("factors of 2",
16                 asList(2), of(2));
17         assertEquals("factors of 3",
18                 asList(3), of(3));
19         assertEquals("factors of 4",
20                 asList(2, 2), of(4));
21         assertEquals("factors of 6",
22                 asList(2, 3), of(6));
23         assertEquals("factors of 8",
24                 asList(2, 2, 2), of(8));
25     }
26 }
27
```

*PrimeFactors.java

```
1  package tdd.primefactors;
2
3  import java.util.ArrayList;
5
6  public class PrimeFactors {
7
8      public static List<Integer> of(int n) {
9          List<Integer> primeFactors = new ArrayList<Integer>();
10         if (n > 1) {
11             if (n % 2 == 0) {
12                 primeFactors.add(2);
13                 n = n/2;
14             }
15             if (n > 1)
16                 primeFactors.add(n);
17         }
18
19         return primeFactors;
20     }
21 }
```

JUnit

Finished after 0.043 seconds

Runs: 1/1 Errors: 0 Failures: 1

tdd.primefactors.PrimeFactorsTestCase [Runner: JUnit 4] (0.018 s)

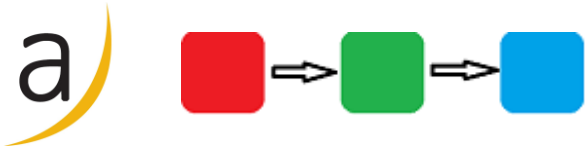
testPrimeFactorsGenerator (0.018 s)

Failure Trace

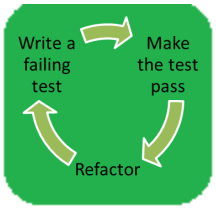
java.lang.AssertionError: factors of 8 expected:<[2, 2, 2]> but was:<[2, 4]>

at tdd.primefactors.PrimeFactorsTestCase.testPrimeFactorsGenerator(Prime

java.lang.AssertionError: factors of 8 expected:<[2, 2, 2]> but was:<[2, 4]>



Prime Factors



PrimeFactorsTestCase.java

```
1  package tdd.primefactors;
2
3  import static java.util.Arrays.asList;
8
9  public class PrimeFactorsTestCase {
10
11     @Test
12     public void testPrimeFactorsGenerator() {
13         assertEquals("factors of 1",
14                 asList(), of(1));
15         assertEquals("factors of 2",
16                 asList(2), of(2));
17         assertEquals("factors of 3",
18                 asList(3), of(3));
19         assertEquals("factors of 4",
20                 asList(2, 2), of(4));
21         assertEquals("factors of 6",
22                 asList(2, 3), of(6));
23         assertEquals("factors of 8",
24                 asList(2, 2, 2), of(8));
25     }
26 }
27
```

PrimeFactors.java

```
1  package tdd.primefactors;
2
3  import java.util.ArrayList;
5
6  public class PrimeFactors {
7      // to fix factors of 8 we replace a "if" with "while"
8      public static List<Integer> of(int n) {
9          List<Integer> primeFactors = new ArrayList<Integer>();
10         if (n > 1) {
11             while (n % 2 == 0) {
12                 primeFactors.add(2);
13                 n = n/2;
14             }
15             if (n > 1)
16                 primeFactors.add(n);
17         }
18
19         return primeFactors;
20     }
21 }
```

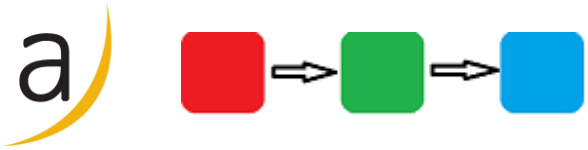
JUnit

Finished after 0.023 seconds

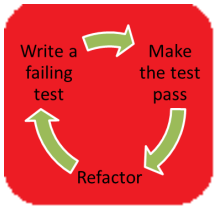
Runs: 1/1 Errors: 0 Failures: 0

tdd.primefactors.PrimeFactorsTestCase [Runner: JUnit 4] (0.001 s)

Failure Trace



Prime Factors



PrimeFactorsTestCase.java

```
1  package tdd.primefactors;
2
3  import static java.util.Arrays.asList;
8
9  public class PrimeFactorsTestCase {
10     // factors of 9
11     @Test
12     public void testPrimeFactorsGenerator() {
13         assertEquals("factors of 1",
14                 asList(), of(1));
15         assertEquals("factors of 2",
16                 asList(2), of(2));
17         assertEquals("factors of 3",
18                 asList(3), of(3));
19         assertEquals("factors of 4",
20                 asList(2, 2), of(4));
21         assertEquals("factors of 6",
22                 asList(2, 3), of(6));
23         assertEquals("factors of 8",
24                 asList(2, 2, 2), of(8));
25         assertEquals("factors of 9",
26                 asList(3, 3), of(9));
27     }
28 }
29
```

PrimeFactors.java

```
1  package tdd.primefactors;
2
3  import java.util.ArrayList;
5
6  public class PrimeFactors {
7
8      public static List<Integer> of(int n) {
9          List<Integer> primeFactors = new ArrayList<Integer>();
10         if (n > 1) {
11             while (n % 2 == 0) {
12                 primeFactors.add(2);
13                 n = n/2;
14             }
15             if (n > 1)
16                 primeFactors.add(n);
17         }
18
19         return primeFactors;
20     }
21 }
```

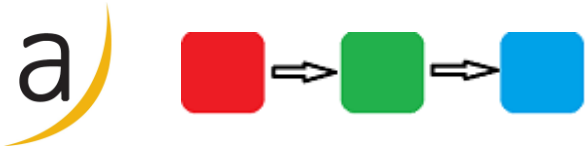
JUnit

Finished after 0.028 seconds

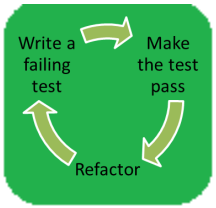
Runs: 1/1 Errors: 0 Failures: 1

tdd.primefactors.PrimeFactorsTestCase [Runner: JUnit 4] (0.005 s)
testPrimeFactorsGenerator (0.005 s)

Failure Trace
java.lang.AssertionError: factors of 9 expected:<[3, 3]> but was:<[9]>
at tdd.primefactors.PrimeFactorsTestCase.testPrimeFactorsGenerator(Prime



Prime Factors



PrimeFactorsTestCase.java

```
1  package tdd.primefactors;
2
3  import static java.util.Arrays.asList;
8
9  public class PrimeFactorsTestCase {
10
11     @Test
12     public void testPrimeFactorsGenerator() {
13         assertEquals("factors of 1",
14                 asList(), of(1));
15         assertEquals("factors of 2",
16                 asList(2), of(2));
17         assertEquals("factors of 3",
18                 asList(3), of(3));
19         assertEquals("factors of 4",
20                 asList(2, 2), of(4));
21         assertEquals("factors of 6",
22                 asList(2, 3), of(6));
23         assertEquals("factors of 8",
24                 asList(2, 2, 2), of(8));
25         assertEquals("factors of 9",
26                 asList(3, 3), of(9));
27     }
28 }
29
```

PrimeFactors.java

```
1  package tdd.primefactors;
2
3  import java.util.ArrayList;
5
6  public class PrimeFactors {
7      // simplest change to fix factors of 9 - is duplicate code
8      public static List<Integer> of(int n) {
9          List<Integer> primeFactors = new ArrayList<Integer>();
10         if (n > 1) {
11             while (n % 2 == 0) {
12                 primeFactors.add(2);
13                 n = n/2;
14             }
15             while (n % 3 == 0) {
16                 primeFactors.add(3);
17                 n = n/3;
18             }
19             if (n > 1)
20                 primeFactors.add(n);
21         }
```

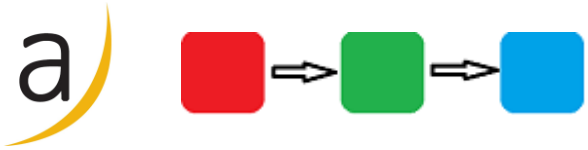
JUnit

Finished after 0.024 seconds

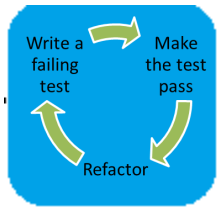
Runs: 1/1 Errors: 0 Failures: 0

tdd.primefactors.PrimeFactorsTestCase [Runner: JUnit 4] (0.000 s)

Failure Trace



Prime Factors



PrimeFactorsTestCase.java

```
1  package tdd.primefactors;
2
3  import static java.util.Arrays.asList;
8
9  public class PrimeFactorsTestCase {
10
11     @Test
12     public void testPrimeFactorsGenerator() {
13         assertEquals("factors of 1",
14                 asList(), of(1));
15         assertEquals("factors of 2",
16                 asList(2), of(2));
17         assertEquals("factors of 3",
18                 asList(3), of(3));
19         assertEquals("factors of 4",
20                 asList(2, 2), of(4));
21         assertEquals("factors of 6",
22                 asList(2, 3), of(6));
23         assertEquals("factors of 8",
24                 asList(2, 2, 2), of(8));
25         assertEquals("factors of 9",
26                 asList(3, 3), of(9));
27     }
28 }
29
```

PrimeFactors.java

```
3  import java.util.ArrayList;
5
6  public class PrimeFactors {
7      // DRY - refactor - first step
8      public static List<Integer> of(int n) {
9          List<Integer> primeFactors = new ArrayList<Integer>();
10         if (n > 1) {
11             int factor = 2;
12             while (factor < n) {
13                 while (n % factor == 0) {
14                     primeFactors.add(factor);
15                     n = n/factor;
16                 }
17                 factor ++;
18             }
19             if (n > 1)
20                 primeFactors.add(n);
21         }
22
23         return primeFactors;
```

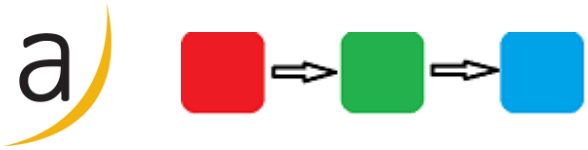
JUnit

Finished after 0.024 seconds

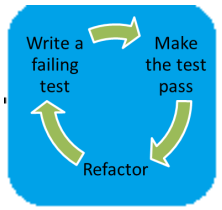
Runs: 1/1 Errors: 0 Failures: 0

tdd.primefactors.PrimeFactorsTestCase [Runner: JUnit 4] (0.000 s)

Failure Trace



Prime Factors



PrimeFactorsTestCase.java

```
1  package tdd.primefactors;
2
3  import static java.util.Arrays.asList;
8
9  public class PrimeFactorsTestCase {
10
11     @Test
12     public void testPrimeFactorsGenerator() {
13         assertEquals("factors of 1",
14                 asList(), of(1));
15         assertEquals("factors of 2",
16                 asList(2), of(2));
17         assertEquals("factors of 3",
18                 asList(3), of(3));
19         assertEquals("factors of 4",
20                 asList(2, 2), of(4));
21         assertEquals("factors of 6",
22                 asList(2, 3), of(6));
23         assertEquals("factors of 8",
24                 asList(2, 2, 2), of(8));
25         assertEquals("factors of 9",
26                 asList(3, 3), of(9));
27     }
28 }
29
```

PrimeFactors.java

```
3  import java.util.ArrayList;
5
6  public class PrimeFactors {
7      // code is ugly - refactor some more, second step
8      public static List<Integer> of(int n) {
9          List<Integer> primeFactors = new ArrayList<Integer>();
10         int factor = 2;
11         while (n > 1) {
12             while (n % factor == 0) {
13                 primeFactors.add(factor);
14                 n = n/factor;
15             }
16             factor ++;
17         }
18
19         if (n > 1)
20             primeFactors.add(n);
21
22         return primeFactors;
23     }
```

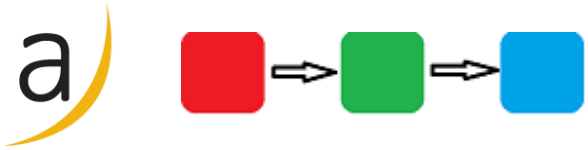
JUnit

Finished after 0.02 seconds

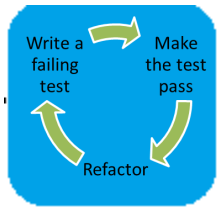
Runs: 1/1 Errors: 0 Failures: 0

tdd.primefactors.PrimeFactorsTestCase [Runner: JUnit 4] (0.000 s)

Failure Trace



Prime Factors



PrimeFactorsTestCase.java

```
1  package tdd.primefactors;
2
3  import static java.util.Arrays.asList;
4
5  public class PrimeFactorsTestCase {
6
7      @Test
8      public void testPrimeFactorsGenerator() {
9          assertEquals("factors of 1",
10                 asList(), of(1));
11         assertEquals("factors of 2",
12                 asList(2), of(2));
13         assertEquals("factors of 3",
14                 asList(3), of(3));
15         assertEquals("factors of 4",
16                 asList(2, 2), of(4));
17         assertEquals("factors of 6",
18                 asList(2, 3), of(6));
19         assertEquals("factors of 8",
20                 asList(2, 2, 2), of(8));
21         assertEquals("factors of 9",
22                 asList(3, 3), of(9));
23     }
24 }
25
```

PrimeFactors.java

```
2
3  import java.util.ArrayList;
4
5  public class PrimeFactors {
6      // refactor - third step - the last if was unnecesary
7      public static List<Integer> of(int n) {
8          List<Integer> primeFactors = new ArrayList<Integer>();
9          int factor = 2;
10         while (n > 1) {
11             while (n % factor == 0) {
12                 primeFactors.add(factor);
13                 n = n/factor;
14             }
15             factor ++;
16         }
17
18         return primeFactors;
19     }
20 }
21
22
```

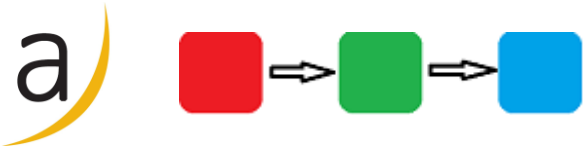
JUnit

Finished after 0.024 seconds

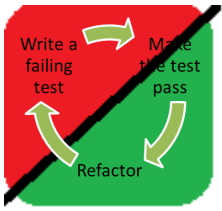
Runs: 1/1 Errors: 0 Failures: 0

tdd.primefactors.PrimeFactorsTestCase [Runner: JUnit 4] (0.000 s)

Failure Trace



Prime Factors



PrimeFactorsTestCase.java

```
1  package tdd.primefactors;
2
3  import static java.util.Arrays.asList;
8
9  public class PrimeFactorsTestCase {
10     // add a new test, factors of 10
11     @Test
12     public void testPrimeFactorsGenerator() {
13         assertEquals("factors of 1",
14                 asList(), of(1));
15         assertEquals("factors of 2",
16                 asList(2), of(2));
17         assertEquals("factors of 3",
18                 asList(3), of(3));
19         assertEquals("factors of 4",
20                 asList(2, 2), of(4));
21         assertEquals("factors of 6",
22                 asList(2, 3), of(6));
23         assertEquals("factors of 8",
24                 asList(2, 2, 2), of(8));
25         assertEquals("factors of 9",
26                 asList(3, 3), of(9));
27         assertEquals("factors of 10",
28                 asList(2, 5), of(10));
29     }
30 }
31
```

PrimeFactors.java

```
2
3  import java.util.ArrayList;
5
6  public class PrimeFactors {
7
8      public static List<Integer> of(int n) {
9          List<Integer> primeFactors = new ArrayList<Integer>();
10         int factor = 2;
11         while (n > 1) {
12             while (n % factor == 0) {
13                 primeFactors.add(factor);
14                 n = n/factor;
15             }
16             factor ++;
17         }
18
19         return primeFactors;
20     }
21 }
22
```

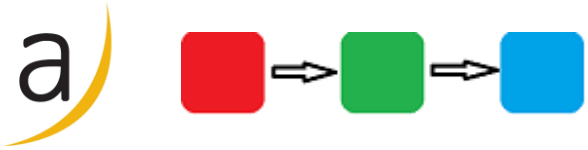
JUnit

Finished after 0.025 seconds

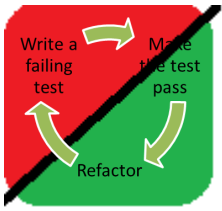
Runs: 1/1 Errors: 0 Failures: 0

tdd.primefactors.PrimeFactorsTestCase [Runner: JUnit 4] (0.000 s)

Failure Trace



Prime Factors



PrimeFactorsTestCase.java

```
 8
 9 public class PrimeFactorsTestCase {
10     // all new tests seem to pass, 11, 12, 18, 1540
11     @Test
12     public void testPrimeFactorsGenerator() {
13         assertEquals("factors of 1",
14                 asList(), of(1));
15         assertEquals("factors of 2",
16                 asList(2), of(2));
17         assertEquals("factors of 3",
18                 asList(3), of(3));
19         assertEquals("factors of 4",
20                 asList(2, 2), of(4));
21         assertEquals("factors of 6",
22                 asList(2, 3), of(6));
23         assertEquals("factors of 8",
24                 asList(2, 2, 2), of(8));
25         assertEquals("factors of 9",
26                 asList(3, 3), of(9));
27         assertEquals("factors of 11",
28                 asList(11), of(11));
29         assertEquals("factors of 12",
30                 asList(2, 2, 3), of(12));
31         assertEquals("factors of 18",
32                 asList(2, 3, 3), of(18));
33         assertEquals("factors of 1540",
34                 asList(2, 2, 5, 7, 11), of(1540));
35     }
```

PrimeFactors.java

```
 2
 3 import java.util.ArrayList;
 5
 6 public class PrimeFactors {
 7
 8     public static List<Integer> of(int n) {
 9         List<Integer> primeFactors = new ArrayList<Integer>();
10         int factor = 2;
11         while (n > 1) {
12             while (n % factor == 0) {
13                 primeFactors.add(factor);
14                 n = n/factor;
15             }
16             factor ++;
17         }
18
19         return primeFactors;
20     }
21 }
22
```

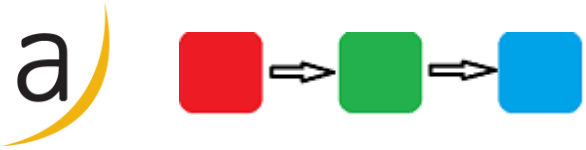
JUnit

Finished after 0.021 seconds

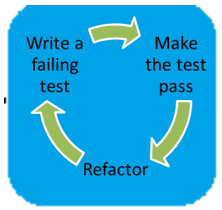
Runs: 1/1 Errors: 0 Failures: 0

tdd.primefactors.PrimeFactorsTestCase [Runner: JUnit 4] (0.000 s)

Failure Trace



Prime Factors



PrimeFactorsTestCase.java

```
 8
 9 public class PrimeFactorsTestCase {
10
11     @Test
12     public void testPrimeFactorsGenerator() {
13         assertEquals("factors of 1",
14                 asList(), of(1));
15         assertEquals("factors of 2",
16                 asList(2), of(2));
17         assertEquals("factors of 3",
18                 asList(3), of(3));
19         assertEquals("factors of 4",
20                 asList(2, 2), of(4));
21         assertEquals("factors of 6",
22                 asList(2, 3), of(6));
23         assertEquals("factors of 8",
24                 asList(2, 2, 2), of(8));
25         assertEquals("factors of 9",
26                 asList(3, 3), of(9));
27         assertEquals("factors of 11",
28                 asList(11), of(11));
29         assertEquals("factors of 12",
30                 asList(2, 2, 3), of(12));
31         assertEquals("factors of 18",
32                 asList(2, 3, 3), of(18));
33         assertEquals("factors of 1540",
34                 asList(2, 2, 5, 7, 11), of(1540));
35     }
```

PrimeFactors.java

```
 1 package tdd.primefactors;
 2
 3 import java.util.ArrayList;
 5
 6 public class PrimeFactors {
 7     // last refactor attempt, cleanup some clutter
 8     public static List<Integer> of(int n) {
 9         List<Integer> primeFactors = new ArrayList<Integer>();
10         for (int factor = 2; n > 1; factor ++)
11             while (n % factor == 0) {
12                 primeFactors.add(factor);
13                 n = n/factor;
14             }
15
16         return primeFactors;
17     }
18 }
19
```

JUnit

Finished after 0.025 seconds

Runs: 1/1 Errors: 0 Failures: 0

tdd.primefactors.PrimeFactorsTestCase [Runner: JUnit 4] (0.001 s)

Failure Trace

Prime Factors

First proposal was:

```
read n
for k = 2 .. n
    if (k is prime)
        copyn = n
        while (copyn % k = 0)
            write k
            copyn = copyn / k
        endwhile
    endif
endfor
end
```

But we ended up with:

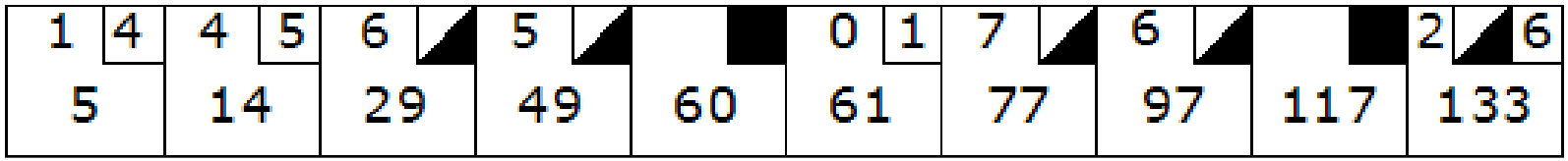
```
read n
for k = 2.. && n > 1
    while (n % k = 0)
        write k
        n = n / k
    endwhile
endfor
end
```

Bowling Score

Bowling Score

10 frames
2 balls per frame

Strike = 10 + next 2 balls
Spare = 10 + next ball
10th frame – max 3 balls;



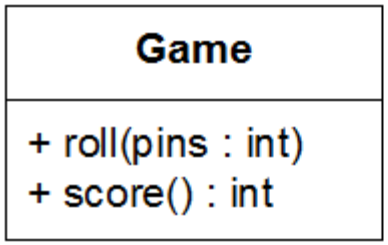
Bowling Score

For the new “Mega Bowl 2000” - you are given the task to develop the “scoring” algorithm. Apart from this, other tasks are for the UI, for the animation, for the player management, etc. – but you are given the most complex task – to compute the total score for a game. Your process needs to be able to *“start”* a new game, record each *“roll”* and at the end of a **correctly played** game it needs to return the *“score”* for that game configuration;

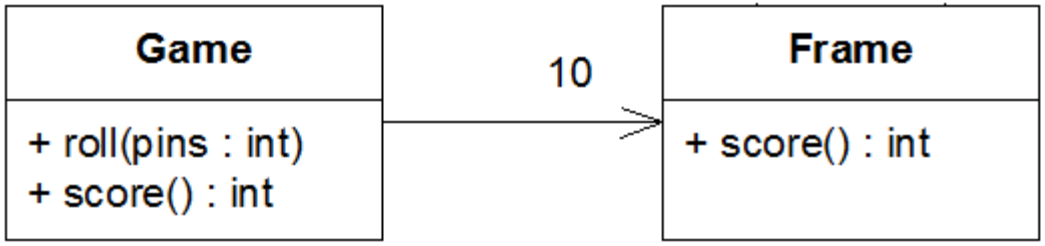
Ex: roll 5, roll 5, roll 2, roll (in order) 0, 10, 0, 4, 0, 0, 8, 1, 10, 1, 3, 9, 0, 3, 7, 3; The total computed score is thus: **81**

Frames																			
1		**2**		**3**	**4**		**5**		**6**		**7**	**8**		**9**		**10**			
5	5	2	0	10	0	4	0	0	8	1	10	1	3	9	0	3	7	3	
Intermediate score																			
12		14		28	32		32		41		55	59		68		81			
spare				**strike**							**strike**					**spare**			
10+2		12+2		14+10+0+4	28+4		32+0		32+9		41+10+1+3	55+4		59+9		68+10+3			

Bowling Score



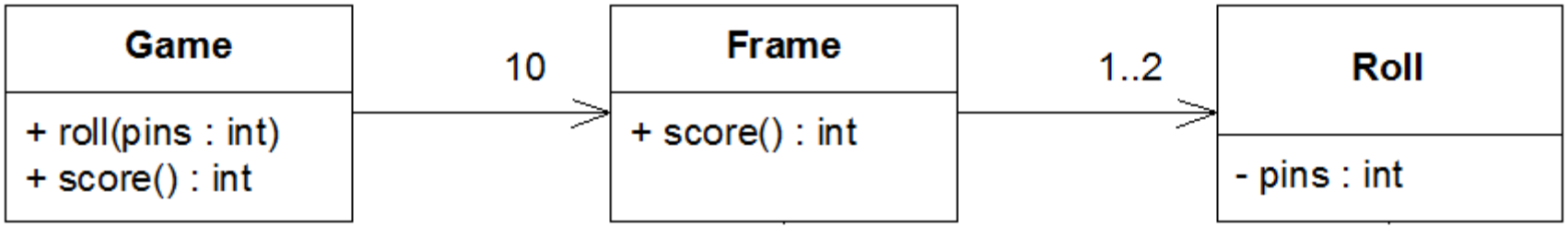
Bowling Score



A game has 10 frames.

Bowling Score

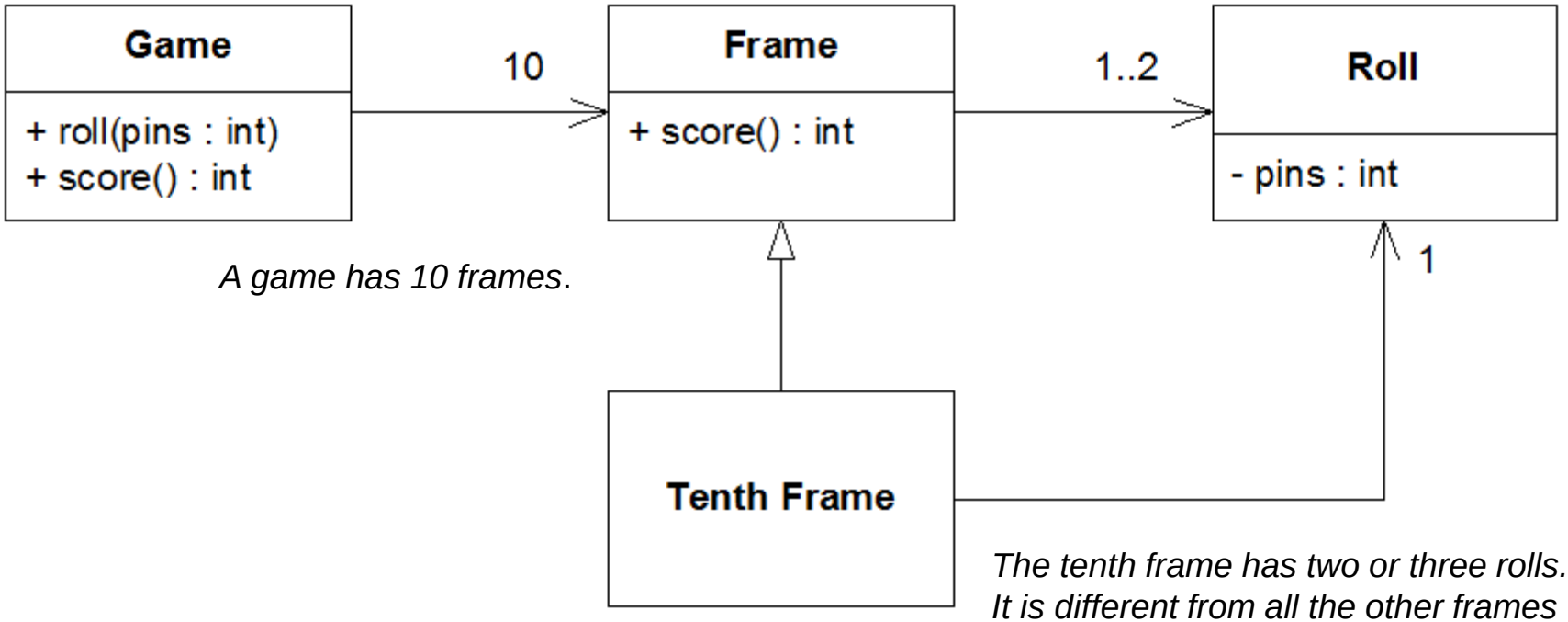
A frame has 1 or two rolls.



A game has 10 frames.

Bowling Score

A frame has 1 or two rolls.

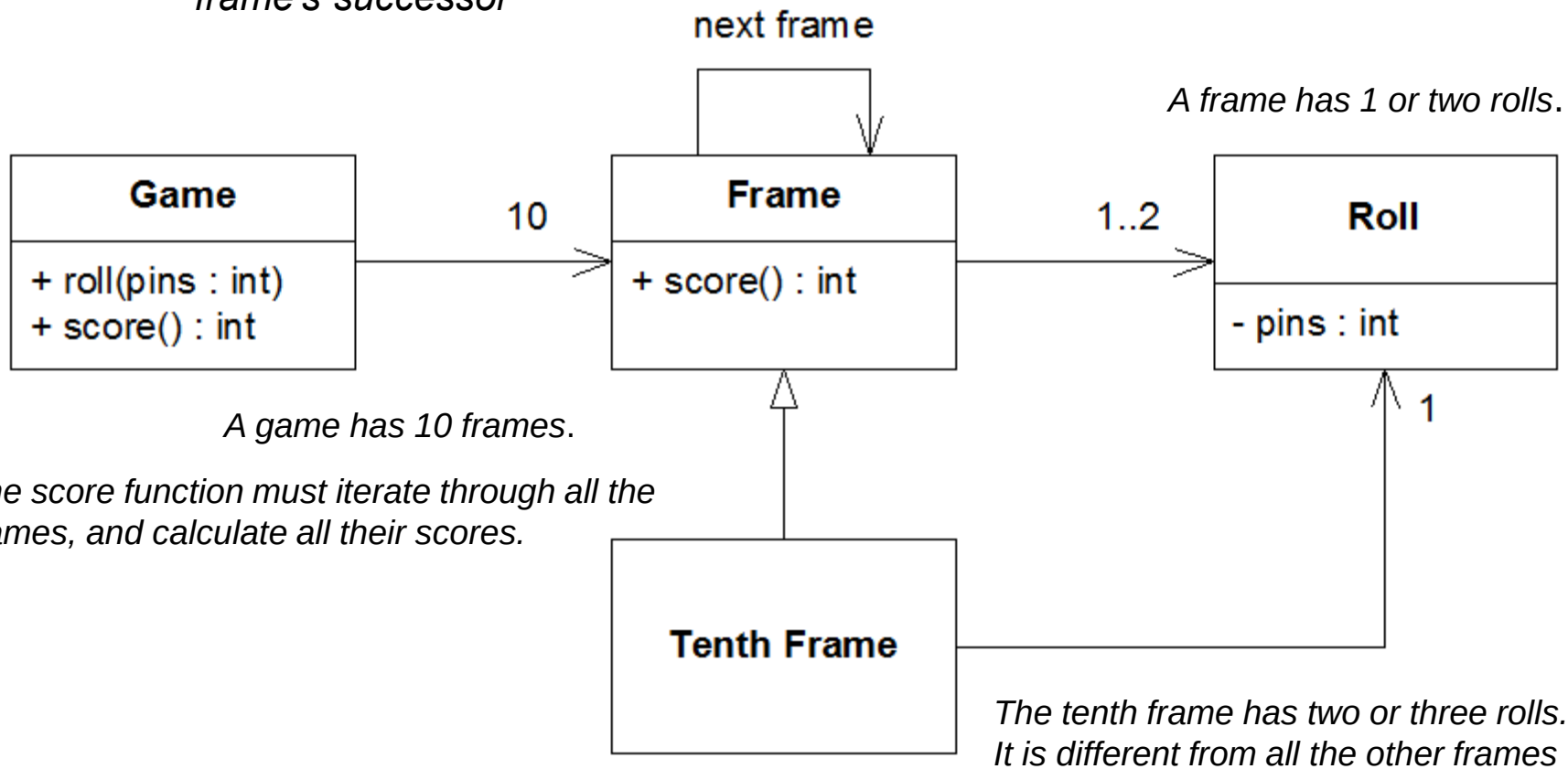


A game has 10 frames.

The tenth frame has two or three rolls.
It is different from all the other frames

Bowling Score

The score for a spare or a strike depends on the frame's successor

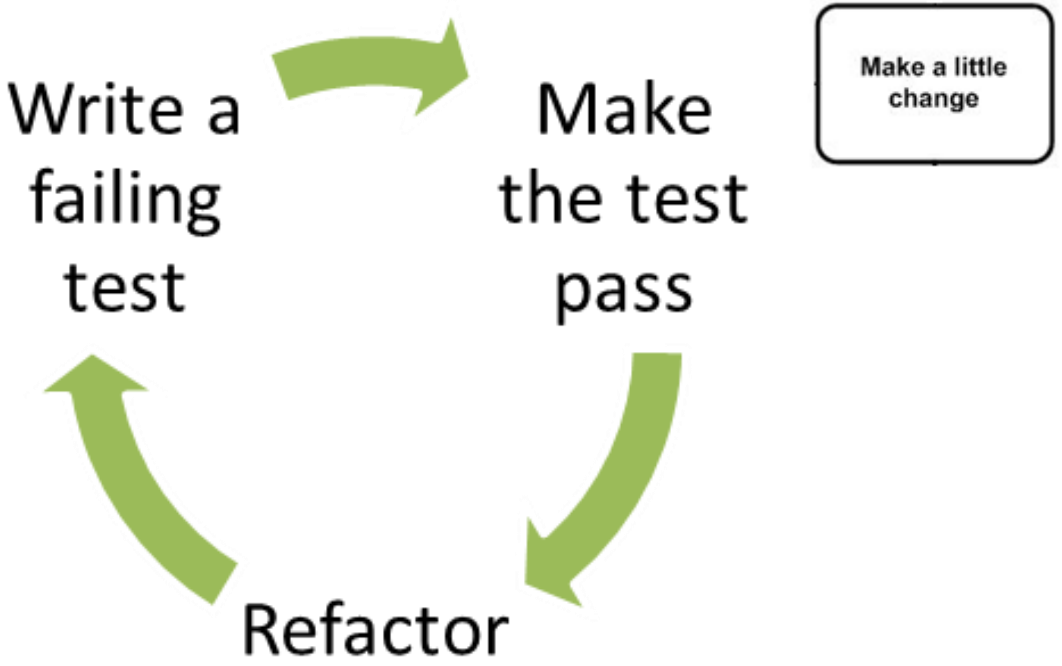


A frame has 1 or two rolls.

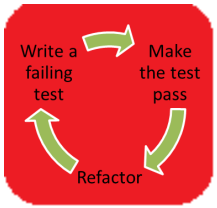
A game has 10 frames.

The score function must iterate through all the frames, and calculate all their scores.

The tenth frame has two or three rolls. It is different from all the other frames



Bowling Score



GameTestCase.java

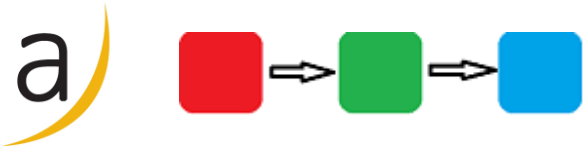
```
1  package tdd.bowlingscore;
2
3  import org.junit.Test;
4
5  public class GameTestCase {
6      // the first test - just create a game
7      @Test
8      public void testStartGame() {
9          Game g = new Game();
10     }
11 }
12
```

Game cannot be resolved to a type

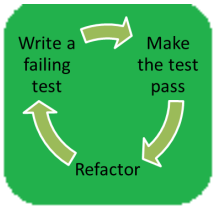
8 quick fixes available:

- Create class 'Game'
- Create interface 'Game'
- Change to 'Game2' (tdd.bowlingscore)
- Change to 'GSSName' (org.ietf.jgss)
- Create enum 'Game'
- Add type parameter 'Game' to 'GameTestCase'
- Add type parameter 'Game' to 'testStartGame()'

Press 'F2' for focus



Bowling Score



GameTestCase.java

```
1  package tdd.bowlingscore;
2
3  import org.junit.Test;
4
5  public class GameTestCase {
6
7      @Test
8      public void testStartGame() {
9          Game g = new Game();
10     }
11 }
12
```

Game.java

```
1  package tdd.bowlingscore;
2
3  public class Game {
4      // fix the test - create the game class
5  }
6
```

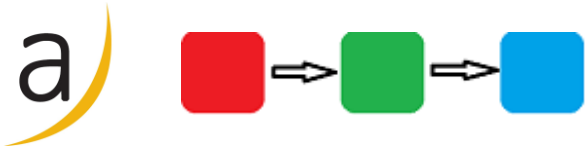
JUnit

Finished after 0.016 seconds

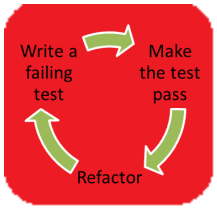
Runs: 1/1 Errors: 0 Failures: 0

tdd.bowlingscore.GameTestCase [Runner: JUnit 4] (0.000 s)

Failure Trace



Bowling Score



GameTestCase.java

```
1  package tdd.bowlingscore;
2
3  import org.junit.Test;
4
5  public class GameTestCase {
6      // second test is to roll a ball
7      @Test
8      public void testStartGame() {
9          Game g = new Game();
10     }
11
12     @Test
13     public void testRoll() {
14         Game g = new Game();
15
16         g.roll(0);
17     }
18 }
19
```

The method roll(int) is undefined for the type Game

2 quick fixes available:

- Create method 'roll(int)' in type 'Game'
- Add cast to 'g'

Press 'F2' for focus

Game.java

```
1  package tdd.bowlingscore;
2
3  public class Game {
4
5  }
6
```

JUnit

Finished after 4.427 seconds

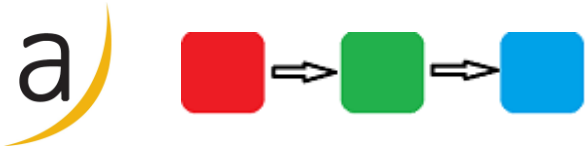
Runs: 2/2 Errors: 1 Failures: 0

tdd.bowlingscore.GameTestCase [Runner: JUnit 4] (4.412 s)
- testStartGame (0.000 s)
- testRoll (4.412 s)

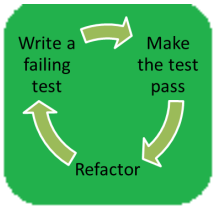
Failure Trace

java.lang.Error: Unresolved compilation problem:
The method roll(int) is undefined for the type Game

at tdd.bowlingscore.GameTestCase.testRoll(GameTestCase.java:16)



Bowling Score



GameTestCase.java

```
1  package tdd.bowlingscore;
2
3  import org.junit.Test;
4
5  public class GameTestCase {
6
7      @Test
8      public void testStartGame() {
9          Game g = new Game();
10     }
11
12     @Test
13     public void testRoll() {
14         Game g = new Game();
15
16         g.roll(0);
17     }
18 }
19
```

Game.java

```
1  package tdd.bowlingscore;
2
3  public class Game {
4
5      // create the roll method -  void with pins argument
6      public void roll(int pins) {
7
8      }
9  }
10
```

JUnit

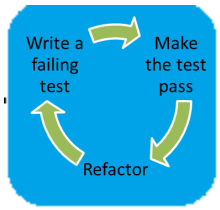
Finished after 0.019 seconds

Runs: 2/2 Errors: 0 Failures: 0

tdd.bowlingscore.GameTestCase [Runner: JUnit 4] (0.000 s)

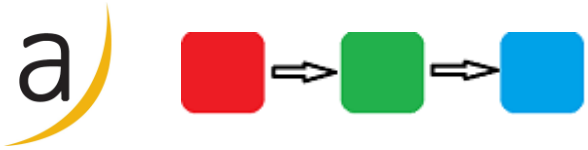
Failure Trace

Bowling Score

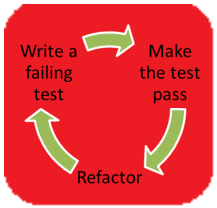



```
 5
 6 // refactor
 7 // each test "sets up" a game
 8 public class GameTestCase {
 9     private Game g;
10
11     @Before
12     public void setUp() {
13         g = new Game();
14     }
15
16     @Test
17     public void testStartGame() {
18     }
19
20     @Test
21     public void testRoll() {
22         g.roll(0);
23     }
```

```
 6 // refactor
 7 // the first test was empty
 8 public class GameTestCase {
 9     private Game g;
10
11     @Before
12     public void setUp() {
13         g = new Game();
14     }
15
16     @Test
17     public void testRoll() {
18         g.roll(0);
19     }
```



Bowling Score



GameTestCase.java

```
1  package tdd.bowlingscore;
2
3  import static org.junit.Assert.assertEquals;
4
5  import org.junit.Before;
6  import org.junit.Test;
7
8  public class GameTestCase {
9      private Game g;
10     // test the score method
11     @Before
12     public void setUp() {
13         g = new Game();
14     }
15
16     @Test
17     public void testRoll() {
18         g.roll(0);
19     }
20
21     @Test
22     public void testScoreGutterGame() {
23         // but in order to "score" we need a full game
24         for (int i = 0; i < 20; i ++)
25             g.roll(0);
26
27         assertEquals(0, g.score());
28     }
```

Game.java

```
1  package tdd.bowlingscore;
2
3  public class Game {
4
5      public void roll(int pins) {
6
7      }
8  }
9
```

JUnit

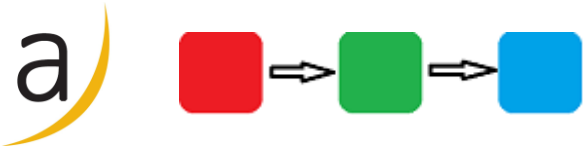
Finished after 1.749 seconds

Runs: 2/2 Errors: 1 Failures: 0

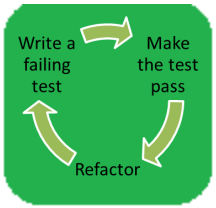
tdd.bowlingscore.GameTestCase [Runner: JUnit 4] (1.711 s)
testScoreGutterGame (1.709 s)
testRoll (0.001 s)

Failure Trace

java.lang.Error: Unresolved compilation problem:
The method score() is undefined for the type Game



Bowling Score



GameTestCase.java

```
1  package tdd.bowlingscore;
2
3  import static org.junit.Assert.assertEquals;
4
5  import org.junit.Before;
6  import org.junit.Test;
7
8  public class GameTestCase {
9      private Game g;
10
11     @Before
12     public void setUp() {
13         g = new Game();
14     }
15
16     @Test
17     public void testRoll() {
18         g.roll(0);
19     }
20
21     @Test
22     public void testScoreGutterGame() {
23         for (int i = 0; i < 20; i ++)
24             g.roll(0);
25
26         assertEquals(0, g.score());
27     }
28 }
```

Game.java

```
1  package tdd.bowlingscore;
2
3  public class Game {
4      // add the score method - fix the compilation
5      public void roll(int pins) {
6      }
7
8      public int score() {
9          return 0;
10     }
11 }
12
```

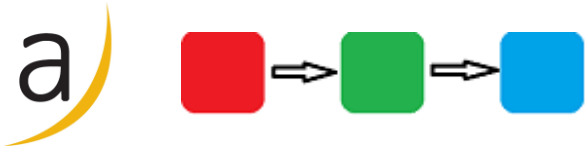
JUnit

Finished after 0.019 seconds

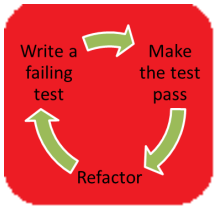
Runs: 2/2 Errors: 0 Failures: 0

tdd.bowlingscore.GameTestCase [Runner: JUnit 4] (0.000 s)

Failure Trace



Bowling Score



GameTestCase.java

```
11    @Before
12    public void setUp() {
13        g = new Game();
14    }
15
16    @Test
17    public void testRoll() {
18        g.roll(0);
19    }
20
21    @Test
22    public void testScoreGutterGame() {
23        for (int i = 0; i < 20; i ++)
24            g.roll(0);
25
26        assertEquals(0, g.score());
27    }
28    // test a game where only 1 is rolled
29    @Test
30    public void testAllOnes() {
31        for (int i = 0; i < 20; i ++) {
32            g.roll(1);
33
34        assertEquals(20, g.score());
35        }
36    }
37 }
38
```

Game.java

```
1  package tdd.bowlingscore;
2
3  public class Game {
4
5      public void roll(int pins) {
6      }
7
8      public int score() {
9          return 0;
10     }
11 }
12
```

JUnit

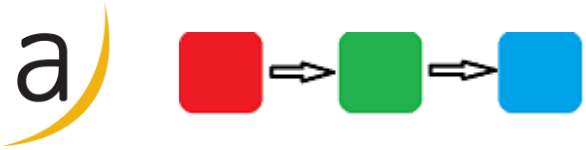
Finished after 0.027 seconds

Runs: 3/3 Errors: 0 Failures: 1

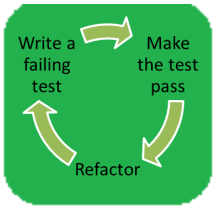
tdd.bowlingscore.GameTestCase [Runner: JUnit 4] (0.005 s)
- testScoreGutterGame (0.000 s)
- testAllOnes (0.004 s)
- testRoll (0.001 s)

Failure Trace

java.lang.AssertionError: expected:<20> but was:<0>
at tdd.bowlingscore.GameTestCase.testAllOnes(GameTestCase.java:



Bowling Score



GameTestCase.java

```
10
11      @Before
12      public void setUp() {
13          g = new Game();
14      }
15
16      @Test
17      public void testRoll() {
18          g.roll(0);
19      }
20
21      @Test
22      public void testScoreGutterGame() {
23          for (int i = 0; i < 20; i ++)
24              g.roll(0);
25
26          assertEquals(0, g.score());
27      }
28
29      @Test
30      public void testAllOnes() {
31          for (int i = 0; i < 20; i ++)
32              g.roll(1);
33
34          assertEquals(20, g.score());
35      }
36  }
37
```

Game.java

```
1  package tdd.bowlingscore;
2
3  public class Game {
4      int score = 0;
5      // actually store the "score" to fix the 1 roll game
6      public void roll(int pins) {
7          score = score + pins;
8      }
9
10     public int score() {
11         return score;
12     }
13 }
14
```

JUnit

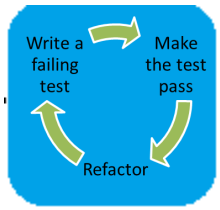
Finished after 0.018 seconds

Runs: 3/3 Errors: 0 Failures: 0

tdd.bowlingscore.GameTestCase [Runner: JUnit 4] (0.000 s)

Failure Trace

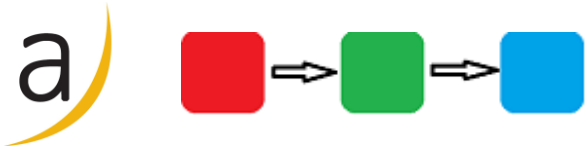
Bowling Score



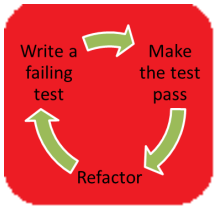

```
// refactor duplicate code
@Test
public void testScoreGutterGame() {
    for (int i = 0; i < 20; i ++)
        g.roll(0);

    assertEquals(0, g.score());
}

@Test
public void testAllOnes() {
    for (int i = 0; i < 20; i ++)
        g.roll(1);

    assertEquals(20, g.score());
}
```

```
@Test
public void testScoreGutterGame() {
    rollMany(0, 20);

    assertEquals(0, g.score());
}

@Test
public void testAllOnes() {
    rollMany(1, 20);

    assertEquals(20, g.score());
}

private void rollMany(int pins,
        int rolls) {
    for (int i = 0; i < rolls; i ++)
        g.roll(pins);
}
```

```
@Test
public void testRoll() {
    g.roll(0); // remove irrelevant test
}
```



Bowling Score



GameTestCase.java

```
16      @Test
17      public void testScoreGutterGame() {
18          rollMany(0, 20);
19          assertEquals(0, g.score());
20      }
21
22      @Test
23      public void testAllOnes() {
24          rollMany(1, 20);
25          assertEquals(20, g.score());
26      }
27      // any valid roll will work
28      // first scenario not covered is to throw a spare
29      @Test
30      public void testSpare() {
31          g.roll(3);
32          g.roll(7); // spare
33          g.roll(4);
34          rollMany(0, 17);
35          assertEquals(18, g.score());
36      }
37
38      private void rollMany(int pins, int rolls) {
39          for (int i = 0; i < rolls; i ++)
40              g.roll(pins);
41      }
42  }
43
```

Game.java

```
1  package tdd.bowlingscore;
2
3  public class Game {
4      int score = 0;
5
6      public void roll(int pins) {
7          score = score + pins;
8      }
9
10     public int score() {
11         return score;
12     }
13 }
14
```

JUnit

Finished after 0.029 seconds

Runs: 3/3 Errors: 0 Failures: 1

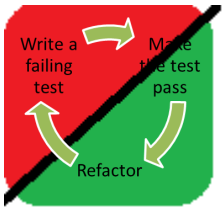
- tdd.bowlingscore.GameTestCase [Runner: JUnit 4] (0.000 s)
  - testScoreGutterGame (0.000 s)
  - testAllOnes (0.000 s)
  - testSpare (0.000 s)

Failure Trace

java.lang.AssertionError: expected:<18> but was:<14>

at tdd.bowlingscore.GameTestCase.testSpare(GameTestCase.java:35)

Bowling Score



GameTestCase.java

```
14      }
15
16      @Test
17      public void testScoreGutterGame() {
18          rollMany(0, 20);
19          assertEquals(0, g.score());
20      }
21
22      @Test
23      public void testAllOnes() {
24          rollMany(1, 20);
25          assertEquals(20, g.score());
26      }
27
28      @Test
29      public void testSpare() {
30          g.roll(3);
31          g.roll(7); // spare
32          g.roll(4);
33          rollMany(0, 17);
34          assertEquals(18, g.score());
35      }
36
37      private void rollMany(int pins, int rolls) {
38          for (int i = 0; i < rolls; i ++)
39              g.roll(pins);
40      }
41  }
```

Game.java

```
1  package tdd.bowlingscore;
2
3  public class Game {
4      int score = 0;
5      // figure out simplest solution to cover spare
6      public void roll(int pins) {
7          // we need to store previous frame score
8          if (previousFrameWasSpare)
9              score = score + pins;
10
11         score = score + pins;
12     }
13     // seems a bit to complicated
14     public int score() {
15         return score;
16     }
17
18     // the 'roll' method computes the score ..
19     // the score just returns the value
20 }
```

JUnit

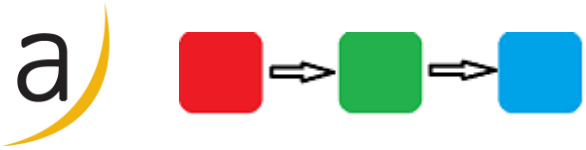
Finished after 0.029 seconds

Runs: 3/3 Errors: 0 Failures: 1

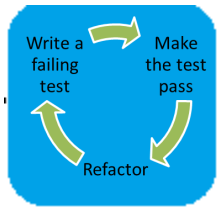
- tdd.bowlingscore.GameTestCase [Runner: JUnit 4] (0.000 s)
  - testScoreGutterGame (0.000 s)
  - testAllOnes (0.000 s)
  - testSpare (0.000 s)

Failure Trace

java.lang.AssertionError: expected:<18> but was:<14>
at tdd.bowlingscore.GameTestCase.testSpare(GameTestCase.java:35)



Bowling Score



GameTestCase.java

```
14      }
15
16      @Test
17      public void testScoreGutterGame() {
18          rollMany(0, 20);
19          assertEquals(0, g.score());
20      }
21
22      @Test
23      public void testAllOnes() {
24          rollMany(1, 20);
25          assertEquals(20, g.score());
26      }
27      // we ignore the new test - so that we can refactor !
28  //  @Test
29      public void testSpare() {
30          g.roll(3);
31          g.roll(7); // spare
32          g.roll(4);
33          rollMany(0, 17);
34          assertEquals(18, g.score());
35      }
36
37      private void rollMany(int pins, int rolls) {
38          for (int i = 0; i < rolls; i ++)
39              g.roll(pins);
40      }
41  }
```

Game.java

```
1  package tdd.bowlingscore;
2  // refactored the code - ALL TESTS still pass
3  public class Game {
4      int[] rolls = new int[21];
5      int currentRoll = 0;
6
7      public void roll(int pins) {
8          rolls[currentRoll ++] = pins;
9      }
10
11     public int score() {
12         int score = 0;
13         for (int i = 0; i < rolls.length; i ++)
14             score += rolls[i];
15         return score;
16     }
17 }
18
```

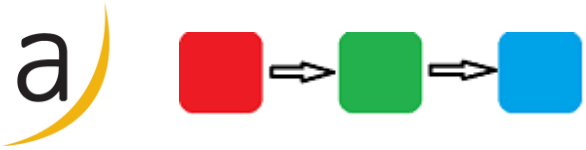
JUnit

Finished after 0.018 seconds

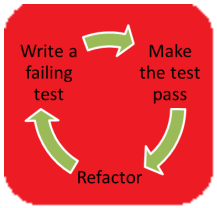
Runs: 2/2 Errors: 0 Failures: 0

tdd.bowlingscore.GameTestCase [Runner: JUnit 4] (0.000 s)

Failure Trace



Bowling Score



GameTestCase.java

```
14      }
15
16      @Test
17      public void testScoreGutterGame() {
18          rollMany(0, 20);
19          assertEquals(0, g.score());
20      }
21
22      @Test
23      public void testAllOnes() {
24          rollMany(1, 20);
25          assertEquals(20, g.score());
26      }
27
28      @Test
29      public void testSpare() {
30          g.roll(3);
31          g.roll(7); // spare
32          g.roll(4);
33          rollMany(0, 17);
34          assertEquals(18, g.score());
35      }
36
37      private void rollMany(int pins, int rolls) {
38          for (int i = 0; i < rolls; i ++)
39              g.roll(pins);
40      }
41  }
```

Game.java

```
1   package tdd.bowlingscore;
2
3   public class Game {
4       int[] rolls = new int[21];
5       int currentRoll = 0;
6
7       public void roll(int pins) {
8           rolls[currentRoll ++] = pins;
9       }
10      // pass the spare test
11      public int score() {
12          int score = 0;
13          for (int i = 0; i < rolls.length; i ++) {
14              if (rolls[i] + rolls[i+1] == 10) // spare
15                  score += rolls[i+2];
16
17              score += rolls[i];
18          } // this attempt is wrong since index if off
19          return score;
20      }
```

JUnit

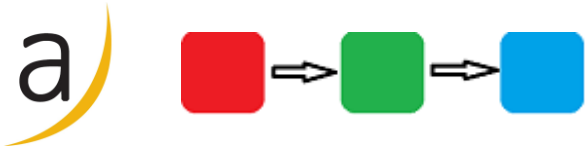
Finished after 0.03 seconds

Runs: 3/3 Errors: 3 Failures: 0

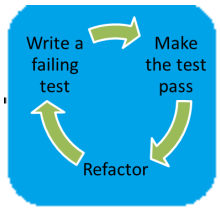
- tdd.bowlingscore.GameTestCase [Runner: JUnit 4] (0.006 s)
  - testScoreGutterGame (0.003 s)
  - testAllOnes (0.000 s)
  - testSpare (0.001 s)

Failure Trace

java.lang.ArrayIndexOutOfBoundsException: 21
at tdd.bowlingscore.Game.score(Game.java:14)
at tdd.bowlingscore.GameTestCase.testScoreGutterGame(GameTest



Bowling Score



GameTestCase.java

```
14      }
15
16      @Test
17      public void testScoreGutterGame() {
18          rollMany(0, 20);
19          assertEquals(0, g.score());
20      }
21
22      @Test
23      public void testAllOnes() {
24          rollMany(1, 20);
25          assertEquals(20, g.score());
26      }
27
28  //  @Test -- comment the test to refactor
29      public void testSpare() {
30          g.roll(3);
31          g.roll(7); // spare
32          g.roll(4);
33          rollMany(0, 17);
34          assertEquals(18, g.score());
35      }
36
37      private void rollMany(int pins, int rolls) {
38          for (int i = 0; i < rolls; i ++)
39              g.roll(pins);
40      }
41  }
```

Game.java

```
1   package tdd.bowlingscore;
2
3   public class Game {
4       int[] rolls = new int[21];
5       int currentRoll = 0;
6
7       public void roll(int pins) {
8           rolls[currentRoll ++] = pins;
9       }
10      // refactor to go through 10 frames, not 21 rolls
11      public int score() {
12          int score = 0;
13          int i = 0;
14          for (int frame = 0; frame < 10; frame ++) {
15              score += rolls[i] + rolls[i+1];
16
17              i += 2;
18          }
19
20          return score;
```

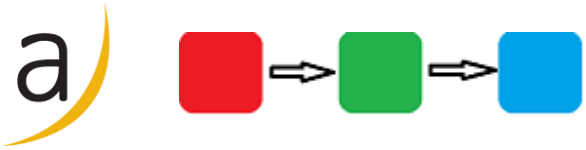
JUnit

Finished after 0.023 seconds

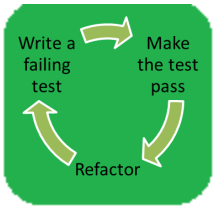
Runs: 2/2 Errors: 0 Failures: 0

tdd.bowlingscore.GameTestCase [Runner: JUnit 4] (0.000 s)

Failure Trace



Bowling Score



GameTestCase.java

```
14      }
15
16      @Test
17      public void testScoreGutterGame() {
18          rollMany(0, 20);
19          assertEquals(0, g.score());
20      }
21
22      @Test
23      public void testAllOnes() {
24          rollMany(1, 20);
25          assertEquals(20, g.score());
26      }
27
28      @Test
29      public void testSpare() {
30          g.roll(3);
31          g.roll(7); // spare
32          g.roll(4);
33          rollMany(0, 17);
34          assertEquals(18, g.score());
35      }
36
37      private void rollMany(int pins, int rolls) {
38          for (int i = 0; i < rolls; i ++)
39              g.roll(pins);
40      }
41  }
```

Game.java

```
3   public class Game {
4       int[] rolls = new int[21];
5       int currentRoll = 0;
6
7       public void roll(int pins) {
8           rolls[currentRoll ++] = pins;
9       }
10      // fix spare test
11      public int score() {
12          int score = 0;
13          int i = 0;
14          for (int frame = 0; frame < 10; frame ++) {
15              if (rolls[i] + rolls[i+1] == 10) // spare
16                  score+= 10 + rolls[i+2];
17              else
18                  score += rolls[i] + rolls[i+1];
19
20              i += 2;
21          }
```

JUnit

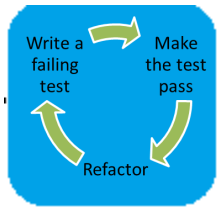
Finished after 0.021 seconds

Runs: 3/3 Errors: 0 Failures: 0

tdd.bowlingscore.GameTestCase [Runner: JUnit 4] (0.001 s)

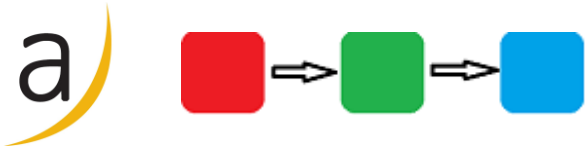
Failure Trace

Bowling Score

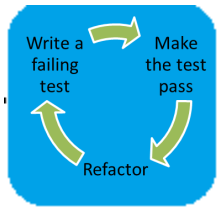



```java
@Test
public void testSpare() {
    g.roll(3);
    g.roll(7); // spare
    g.roll(4);
    rollMany(0, 17);
    assertEquals(18, g.score());
}
```

```java
@Test
public void testSpare() {
    rollSpare();
    g.roll(4);
    rollMany(0, 17);
    assertEquals(18, g.score());
}
// if you need inline comment to explain something
// your code is not clear
private void rollSpare() {
    g.roll(3);
    g.roll(7);
}
```



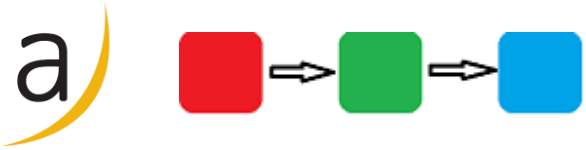
Bowling Score



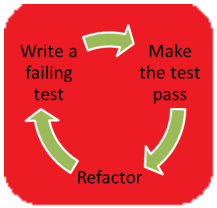

```
int i = 0;
for (int frame = 0; frame < 10; frame ++) {
    if (rolls[i] + rolls[i+1] == 10) // spare
        score+= 10 + rolls[i+2];
    else
        score += rolls[i] + rolls[i+1];

    i += 2;
}
```

```
int rollIndex = 0;
for (int frame = 0; frame < 10; frame ++) {
    if (isSpare(rollIndex))
        score+= 10 + rolls[rollIndex+2];
    else
        score += rolls[rollIndex] + rolls[rollIndex+1];

    rollIndex += 2;
}
```

```
private boolean isSpare(int rollIndex) {
    return rolls[rollIndex] + rolls[rollIndex+1] == 10;
}
```



Bowling Score



GameTestCase.java

```
22      @Test
23      public void testAllOnes() {
24          rollMany(1, 20);
25          assertEquals(20, g.score());
26      }
27
28      @Test
29      public void testSpare() {
30          rollSpare();
31          g.roll(4);
32          rollMany(0, 17);
33          assertEquals(18, g.score());
34      }
35
36      @Test
37      public void testStrike() {
38          g.roll(10); // strike
39          g.roll(3);
40          g.roll(4);
41          rollMany(0, 17);
42          assertEquals(24, g.score());
43      }
44
45      private void rollSpare() {
46          g.roll(3);
47          g.roll(7);
48      }
49
```

Game.java

```
11      public int score() {
12          int score = 0;
13          int rollIndex = 0;
14          for (int frame = 0; frame < 10; frame ++) {
15              if (isSpare(rollIndex))
16                  score+= 10 + rolls[rollIndex+2];
17              else
18                  score += rolls[rollIndex] + rolls[rollIndex+1];
19
20              rollIndex += 2;
21          }
22
23          return score;
24      }
25
26      private boolean isSpare(int rollIndex) {
27          return rolls[rollIndex] + rolls[rollIndex+1] == 10;
28      }
29  }
```

JUnit

Finished after 0.025 seconds

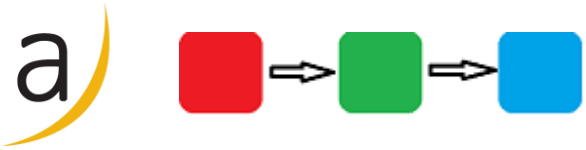
Runs: 4/4 Errors: 0 Failures: 1

- tdd.bowlingscore.GameTestCase [Runner: JUnit 4] (0.000 s)
  - testScoreGutterGame (0.000 s)
  - testAllOnes (0.000 s)
  - testSpare (0.000 s)
  - testStrike (0.000 s)

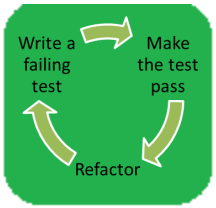
Failure Trace

java.lang.AssertionError: expected:<24> but was:<17>

at tdd.bowlingscore.GameTestCase.testStrike(GameTestCase.java:42)



Bowling Score



GameTestCase.java

```
17      public void testScoreGutterGame() {
18          rollMany(0, 20);
19          assertEquals(0, g.score());
20      }
21
22      @Test
23      public void testAllOnes() {
24          rollMany(1, 20);
25          assertEquals(20, g.score());
26      }
27
28      @Test
29      public void testSpare() {
30          rollSpare();
31          g.roll(4);
32          rollMany(0, 17);
33          assertEquals(18, g.score());
34      }
35
36      @Test
37      public void testStrike() {
38          g.roll(10); // strike
39          g.roll(3);
40          g.roll(4);
41          rollMany(0, 17);
42          assertEquals(24, g.score());
43      }
44
```

Game.java

```
11  public int score() {
12      int score = 0;
13      int rollIndex = 0;
14      for (int frame = 0; frame < 10; frame ++) {
15          if (rolls[rollIndex] == 10) { // strike
16              score += 10 + rolls[rollIndex+1] + rolls[rollIndex+2];
17              rollIndex ++;
18          } else {
19              if (isSpare(rollIndex))
20                  score += 10 + rolls[rollIndex+2];
21              else
22                  score += rolls[rollIndex] + rolls[rollIndex+1];
23
24              rollIndex += 2;
25          }
26      }
27
28      return score;
29  }
```

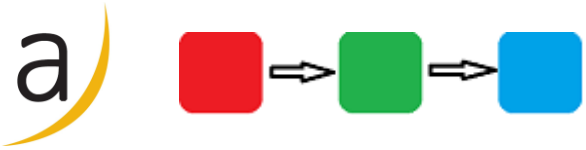
JUnit

Finished after 0.019 seconds

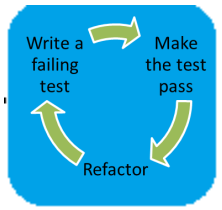
Runs: 4/4 Errors: 0 Failures: 0

tdd.bowlingscore.GameTestCase [Runner: JUnit 4] (0.000 s)

Failure Trace



Bowling Score



GameTestCase.java

```
21
22    @Test
23    public void testAllOnes() {
24        rollMany(1, 20);
25        assertEquals(20, g.score());
26    }
27
28    @Test
29    public void testSpare() {
30        rollSpare();
31        g.roll(4);
32        rollMany(0, 17);
33        assertEquals(18, g.score());
34    }
35
36    @Test
37    public void testStrike() {
38        rollStrike();
39        g.roll(3);
40        g.roll(4);
41        rollMany(0, 17);
42        assertEquals(24, g.score());
43    }
44
45    private void rollSpare() {
46        g.roll(3);
47        g.roll(7);
48    }
49
50    private void rollStrike() {
51        g.roll(10);
52    }
53
54    private void rollMany(int pins, int rolls) {
55        for (int i = 0; i < rolls; i ++)
56            g.roll(pins);
```

Game.java

```
10
11    public int score() {
12        int score = 0;
13        int rollIndex = 0;
14        for (int frame = 0; frame < 10; frame ++) {
15            if (isStrike(rollIndex)) {
16                score += 10 + rolls[rollIndex+1] + rolls[rollIndex+2];
17                rollIndex ++;
18            } else {
19                if (isSpare(rollIndex))
20                    score += 10 + rolls[rollIndex+2];
21                else
22                    score += rolls[rollIndex] + rolls[rollIndex+1];
23
24                rollIndex += 2;
25            }
26        }
27
28        return score;
29    }
30
31    private boolean isStrike(int rollIndex) {
32        return rolls[rollIndex] == 10;
33    }
34
```

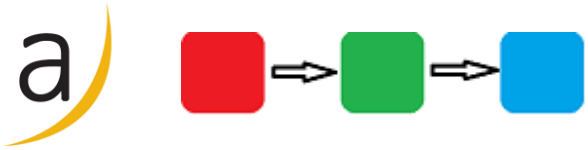
JUnit

Finished after 0.019 seconds

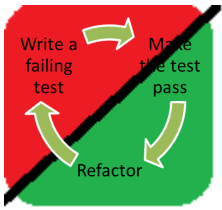
Runs: 4/4 Errors: 0 Failures: 0

tdd.bowlingscore.GameTestCase [Runner: JUnit 4] (0.000 s)

Failure Trace



Bowling Score



GameTestCase.java

```
21
22      @Test
23      public void testAllOnes() {
24          rollMany(1, 20);
25          assertEquals(20, g.score());
26      }
27
28      @Test
29      public void testSpare() {
30          rollSpare();
31          g.roll(4);
32          rollMany(0, 17);
33          assertEquals(18, g.score());
34      }
35
36      @Test
37      public void testStrike() {
38          rollStrike();
39          g.roll(3);
40          g.roll(4);
41          rollMany(0, 17);
42          assertEquals(24, g.score());
43      }
44
45      // any other spare + strike scenario tests seem to pass
46      @Test
47      public void testPerfectGame() {
48          rollMany(10, 12);
49          assertEquals(300, g.score());
50      }
51
52      private void rollSpare() {
53          g.roll(3);
54          g.roll(7);
55      }
```

Game.java

```
10
11      public int score() {
12          int score = 0;
13          int rollIndex = 0;
14          for (int frame = 0; frame < 10; frame ++) {
15              if (isStrike(rollIndex)) {
16                  score += 10 + rolls[rollIndex+1] + rolls[rollIndex+2];
17                  rollIndex ++;
18              } else {
19                  if (isSpare(rollIndex))
20                      score += 10 + rolls[rollIndex+2];
21                  else
22                      score += rolls[rollIndex] + rolls[rollIndex+1];
23
24                  rollIndex += 2;
25              }
26          }
27
28          return score;
29      }
30
31      private boolean isStrike(int rollIndex) {
32          return rolls[rollIndex] == 10;
33      }
34
```

JUnit

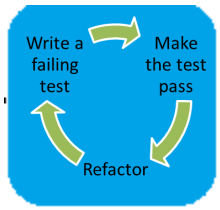
Finished after 0.018 seconds

Runs: 5/5 Errors: 0 Failures: 0

tdd.bowlingscore.GameTestCase [Runner: JUnit 4] (0.000 s)

Failure Trace

Bowling Score




```
public int score() {
    int score = 0;
    int rollIndex = 0;
    for (int frame = 0; frame < 10; frame ++) {
        if (isStrike(rollIndex)) {
            score += 10 + strikeBonus(rollIndex);
            rollIndex ++;
        } else {
            if (isSpare(rollIndex))
                score += 10 + spareBonus(rollIndex);
            else
                score += sumOfFrameRolls(rollIndex);

            rollIndex += 2;
        }
    }

    return score;
}
```

```
private boolean isStrike(int rollIndex) {
    return rolls[rollIndex] == 10;
}
```

```
private int strikeBonus(int rollIndex) {
    return rolls[rollIndex+1] + rolls[rollIndex+2];
}
```

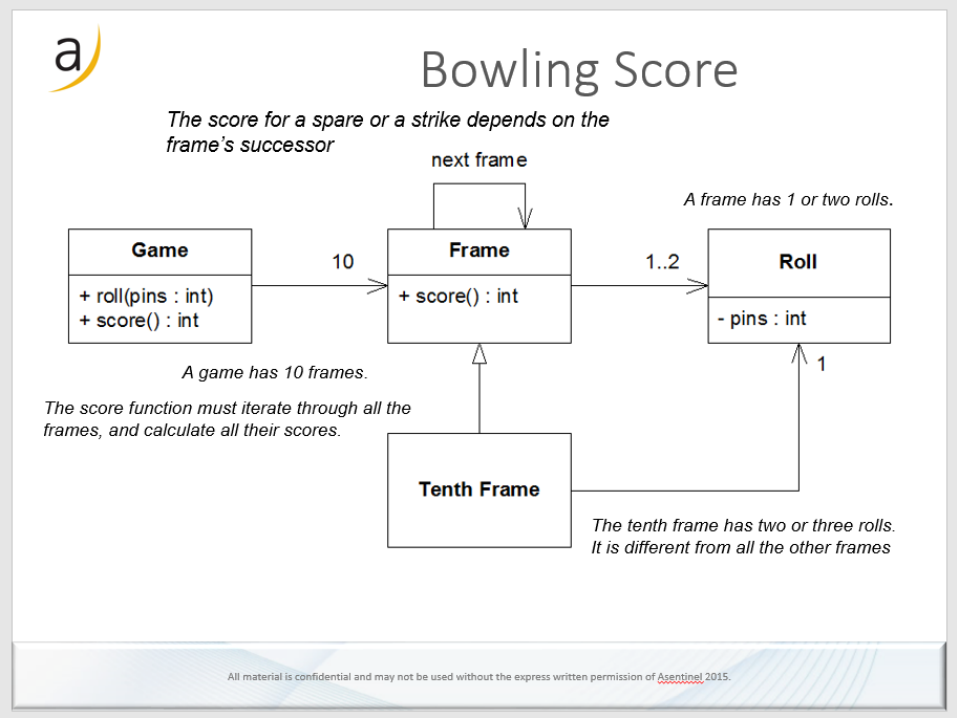
```
private boolean isSpare(int rollIndex) {
    return rolls[rollIndex] + rolls[rollIndex+1] == 10;
}
```

```
private int spareBonus(int rollIndex) {
    return rolls[rollIndex+2];
}
```

```
private int sumOfFrameRolls(int rollIndex) {
    return rolls[rollIndex] + rolls[rollIndex+1];
}
```

Bowling Score

Instead of :



We simply have:

```
for frames : 1 .. 10
        if frame is strike
                score += 10 + next 2 balls
                skip 1 ball
        else
                if frame is spare
                        score += 10 + next ball
                else
                        score += balls in frame
                end if
                skip 2 balls
        end if
end for
```

Roman Numerals

Roman Numerals

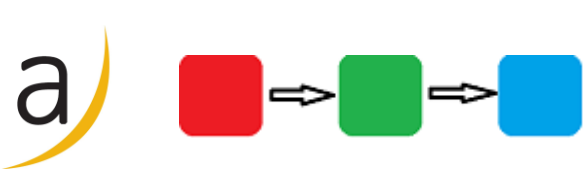
Roman numeric system

Roman numerals, as used today, are based on seven symbols:[1]

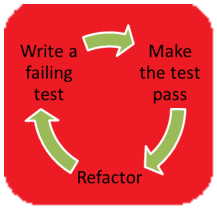
Symbol	Value
I	1
V	5
X	10
L	50
C	100
D	500
M	1,000

- I placed before V or X indicates one less, so four is IV (one less than five) and nine is IX (one less than ten)
- X placed before L or C indicates ten less, so forty is XL (ten less than fifty) and ninety is XC (ten less than a hundred)
- C placed before D or M indicates a hundred less, so four hundred is CD (a hundred less than five hundred) and nine hundred is CM (a hundred less than a thousand)[5]

789 – DCCLXXXIX
1234 – MCCXXXIV
99 – XCIX



Roman Numerals



RomanNumeralsTestCase.java

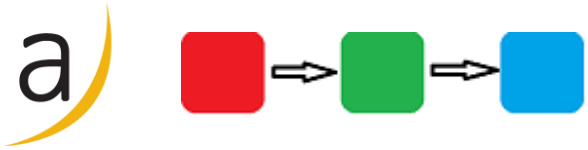
```
1  package tdd.romannumerals;
2
3  import static org.junit.Assert.assertEquals;
4
5  import org.junit.Test;
6
7  public class RomanNumeralsTestCase {
8      @Test
9      public void testOneIsTransformed() {
10         assertEquals("I", RomanNumerals.of(1));
11     }
12 }
13
```

RomanNumerals cannot be resolved

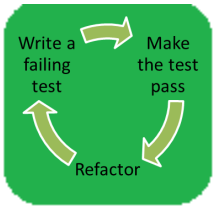
10 quick fixes available:

- Create class 'RomanNumerals'
- Create interface 'RomanNumerals'
- Create constant 'RomanNumerals'
- Create enum 'RomanNumerals'
- Create local variable 'RomanNumerals'
- Change to 'RomanNumeralsTestCase' (tdd.romannumerals)
- Create field 'RomanNumerals'

Press 'F2' for focus



Roman Numerals



RomanNumeralsTestCase.java

```
1  package tdd.romannumerals;
2
3  import static org.junit.Assert.assertEquals;
4
5  import org.junit.Test;
6
7  public class RomanNumeralsTestCase {
8
9      @Test
10     public void testOneIsTransformed() {
11         assertEquals("I", RomanNumerals.of(1));
12     }
13
14 }
15
```

RomanNumerals.java

```
1  package tdd.romannumerals;
2
3  public class RomanNumerals {
4
5      public static String of(int value) {
6          return "I";
7      }
8
9  }
10
```

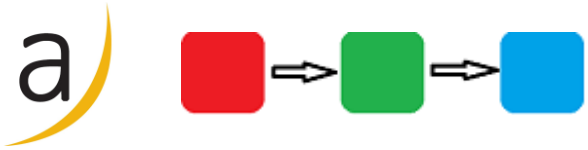
JUnit

Finished after 0.027 seconds

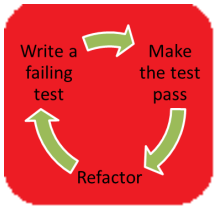
Runs: 1/1 Errors: 0 Failures: 0

tdd.romannumerals.RomanNumeralsTestCase [Runner: JUnit 4] (0.

Failure Trace



Roman Numerals



RomanNumeralsTestCase.java

```java
package tdd.romannumerals;

import static org.junit.Assert.assertEquals;

import org.junit.Test;

public class RomanNumeralsTestCase {

    @Test
    public void testOneIsTransformed() {
        assertEquals("I", RomanNumerals.of(1));
    }

    @Test // we could have as easily chose "5" as the next test
    public void testTwoIsTransformed() {
        assertEquals("II", RomanNumerals.of(2));
    }

}
```

RomanNumerals.java

```java
package tdd.romannumerals;

public class RomanNumerals {

    public static String of(int value) {
        return "I";
    }

}
```

JUnit

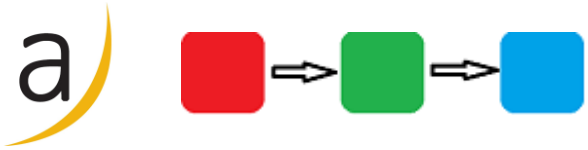
Finished after 0.074 seconds

Runs: 2/2 Errors: 0 Failures: 1

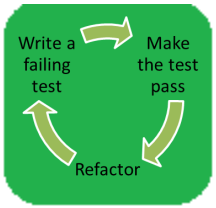
testOneIsTransformed (0.000 s)
testTwoIsTransformed (0.051 s)

Failure Trace

org.junit.ComparisonFailure: expected:<I[I]> but was:<I[]>



Roman Numerals



RomanNumeralsTestCase.java

```
1  package tdd.romannumerals;
2
3  import static org.junit.Assert.assertEquals;
4
5  import org.junit.Test;
6
7  public class RomanNumeralsTestCase {
8
9      @Test
10     public void testOneIsTransformed() {
11         assertEquals("I", RomanNumerals.of(1));
12     }
13
14     @Test
15     public void testTwoIsTransformed() {
16         assertEquals("II", RomanNumerals.of(2));
17     }
18
19 }
20
```

RomanNumerals.java

```
1  package tdd.romannumerals;
2
3  public class RomanNumerals {
4
5      public static String of(int value) {
6          if (value == 2)
7              return "II";
8
9          return "I";
10     }
11
12 }
13
```

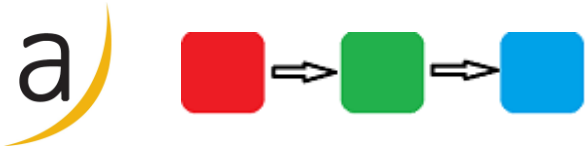
JUnit

Finished after 0.022 seconds

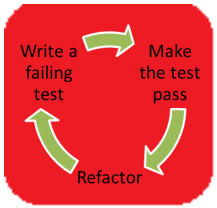
Runs: 2/2 Errors: 0 Failures: 0

tdd.romannumerals.RomanNumeralsTestCase [Runner: JUnit 4] (0.0

Failure Trace



Roman Numerals



RomanNumeralsTestCase.java

```
1  package tdd.romannumerals;
2
3  import static org.junit.Assert.assertEquals;
4
5  import org.junit.Test;
6
7  public class RomanNumeralsTestCase {
8
9      @Test
10     public void testOneIsTransformed() {
11         assertEquals("I", RomanNumerals.of(1));
12     }
13
14     @Test
15     public void testTwoIsTransformed() {
16         assertEquals("II", RomanNumerals.of(2));
17     }
18
19     @Test
20     public void testThreeIsTransformed() {
21         assertEquals("III", RomanNumerals.of(3));
22     }
23
24 }
25
```

RomanNumerals.java

```
1  package tdd.romannumerals;
2
3  public class RomanNumerals {
4
5      public static String of(int value) {
6          if (value == 2)
7              return "II";
8
9          return "I";
10     }
11
12 }
13
```

JUnit

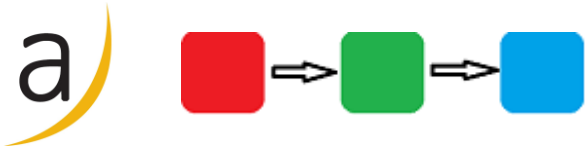
Finished after 0.031 seconds

Runs: 3/3 Errors: 0 Failures: 1

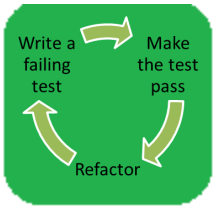
testOneIsTransformed (0.000 s)
testThreeIsTransformed (0.009 s)
testTwoIsTransformed (0.000 s)

Failure Trace

org.junit.ComparisonFailure: expected:<I[II]> but was:<I[]>



Roman Numerals



RomanNumeralsTestCase.java

```java
package tdd.romannumerals;

import static org.junit.Assert.assertEquals;

import org.junit.Test;

public class RomanNumeralsTestCase {

    @Test
    public void testOneIsTransformed() {
        assertEquals("I", RomanNumerals.of(1));
    }

    @Test
    public void testTwoIsTransformed() {
        assertEquals("II", RomanNumerals.of(2));
    }

    @Test
    public void testThreeIsTransformed() {
        assertEquals("III", RomanNumerals.of(3));
    }

}
```

RomanNumerals.java

```java
package tdd.romannumerals;

public class RomanNumerals {

    public static String of(int value) {
        if (value == 3) // we will definetly refactor at the next step
            return "III";

        if (value == 2)
            return "II";

        return "I";
    }

}
```

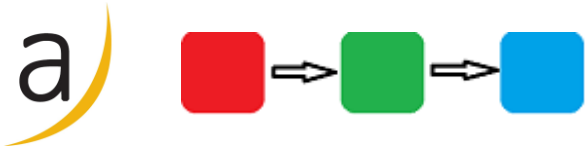
JUnit

Finished after 0.022 seconds

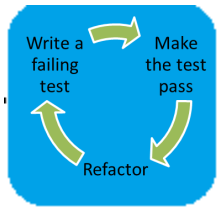
Runs: 3/3 Errors: 0 Failures: 0

tdd.romannumerals.RomanNumeralsTestCase [Runner: JUnit 4] (0.

Failure Trace



Roman Numerals



RomanNumeralsTestCase.java

```
1  package tdd.romannumerals;
2
3  import static org.junit.Assert.assertEquals;
4
5  import org.junit.Test;
6
7  public class RomanNumeralsTestCase {
8
9      @Test
10     public void testOneIsTransformed() {
11         assertEquals("I", RomanNumerals.of(1));
12     }
13
14     @Test
15     public void testTwoIsTransformed() {
16         assertEquals("II", RomanNumerals.of(2));
17     }
18
19     @Test
20     public void testThreeIsTransformed() {
21         assertEquals("III", RomanNumerals.of(3));
22     }
23
24 }
25
```

RomanNumerals.java

```
1  package tdd.romannumerals;
2
3  public class RomanNumerals {
4
5      public static String of(int value) {
6          String romanValue = "";
7
8          while (value -- > 0)
9              romanValue += "I";
10
11         return romanValue;
12     }
13
14 }
15
```

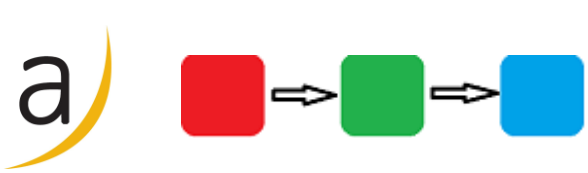
JUnit

Finished after 0.024 seconds

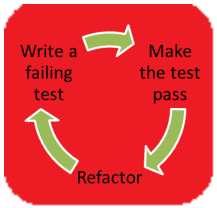
Runs: 3/3 Errors: 0 Failures: 0

tdd.romannumerals.RomanNumeralsTestCase [Runner: JUnit 4] (0.0

Failure Trace



Roman Numerals



RomanNumeralsTestCase.java

```
7  public class RomanNumeralsTestCase {
8
9      @Test
10     public void testOne() {
11         assertEquals("I", RomanNumerals.of(1));
12     }
13
14     @Test
15     public void testTwo() {
16         assertEquals("II", RomanNumerals.of(2));
17     }
18
19     @Test
20     public void testThree() {
21         assertEquals("III", RomanNumerals.of(3));
22     }
23
24     // should we verify the 5 or the 10 ?
25     // we can have X / XX / XXX - but not V / VV / VVV
26     // let's work the pattern
27     @Test
28     public void testTen() {
29         assertEquals("X", RomanNumerals.of(10));
30     }
31 }
32
```

RomanNumerals.java

```
1  package tdd.romannumerals;
2
3  public class RomanNumerals {
4
5      public static String of(int value) {
6          String romanValue = "";
7
8          while (value -- > 0)
9              romanValue += "I";
10
11         return romanValue;
12     }
13
14 }
15
```

JUnit

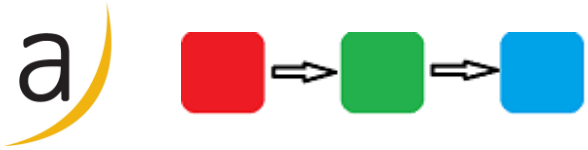
Finished after 0.038 seconds

Runs: 4/4 Errors: 0 Failures: 1

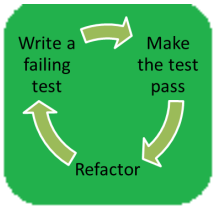
testOne (0.000 s)
testTen (0.012 s)
testTwo (0.000 s)

Failure Trace

org.junit.ComparisonFailure: expected:<[X]> but was:<[IIIIIIIIII]>



Roman Numerals



RomanNumeralsTestCase.java

```
public class RomanNumeralsTestCase {

    @Test
    public void testOne() {
        assertEquals("I", RomanNumerals.of(1));
    }

    @Test
    public void testTwo() {
        assertEquals("II", RomanNumerals.of(2));
    }

    @Test
    public void testThree() {
        assertEquals("III", RomanNumerals.of(3));
    }

    // should we verify the 5 or the 10 ?
    // we can have X / XX / XXX - but not V / VV / VVV
    // let's work the pattern
    @Test
    public void testTen() {
        assertEquals("X", RomanNumerals.of(10));
    }
}
```

RomanNumerals.java

```
package tdd.romannumerals;

public class RomanNumerals {

    public static String of(int value) {
        String romanValue = "";

        if (value == 10)
            return "X";

        while (value -- > 0)
            romanValue += "I";

        return romanValue;
    }

}
```

JUnit

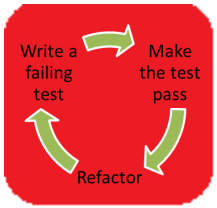
Finished after 0.023 seconds

Runs: 4/4 Errors: 0 Failures: 0

tdd.romannumerals.RomanNumeralsTestCase [Runner: JUnit 4] (0.0

Failure Trace

Roman Numerals



RomanNumeralsTestCase.java

```
    @Test
    public void testOne() {
        assertEquals("I", RomanNumerals.of(1));
    }

    @Test
    public void testTwo() {
        assertEquals("II", RomanNumerals.of(2));
    }

    @Test
    public void testThree() {
        assertEquals("III", RomanNumerals.of(3));
    }

    @Test
    public void testTen() {
        assertEquals("X", RomanNumerals.of(10));
    }

    @Test
    public void testTwenty() {
        assertEquals("XX", RomanNumerals.of(20));
    }
}
```

RomanNumerals.java

```
1  package tdd.romannumerals;
2
3  public class RomanNumerals {
4
5      public static String of(int value) {
6          String romanValue = "";
7
8          if (value == 10)
9              return "X";
10
11         while (value -- > 0)
12             romanValue += "I";
13
14         return romanValue;
15     }
16
17 }
18
```

JUnit

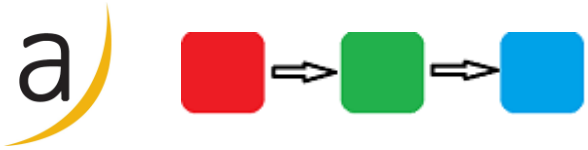
Finished after 0.037 seconds

Runs: 5/5 Errors: 0 Failures: 1

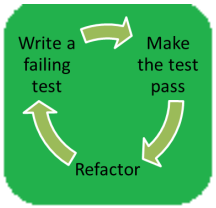
testTwo (0.000 s)
testThree (0.000 s)
testTwenty (0.009 s)

Failure Trace

org.junit.ComparisonFailure: expected:<[XX]> but was:<[IIIIIIIIIIIIIIIIII



Roman Numerals



RomanNumeralsTestCase.java

```
    @Test
    public void testOne() {
        assertEquals("I", RomanNumerals.of(1));
    }

    @Test
    public void testTwo() {
        assertEquals("II", RomanNumerals.of(2));
    }

    @Test
    public void testThree() {
        assertEquals("III", RomanNumerals.of(3));
    }

    @Test
    public void testTen() {
        assertEquals("X", RomanNumerals.of(10));
    }

    @Test
    public void testTwenty() {
        assertEquals("XX", RomanNumerals.of(20));
    }
}
```

RomanNumerals.java

```
package tdd.romannumerals;

public class RomanNumerals {

    public static String of(int value) {
        String romanValue = "";
        // we should refactor this later - the tests pass now
        while (value >= 10) {
            romanValue += "X";
            value -= 10;
        }

        while (value -- > 0)
            romanValue += "I";

        return romanValue;
    }

}
```

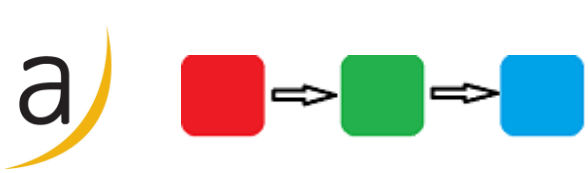
JUnit

Finished after 0.024 seconds

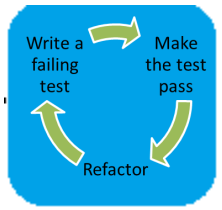
Runs: 5/5 Errors: 0 Failures: 0

tdd.romannumerals.RomanNumeralsTestCase [Runner: JUnit 4] (0.0

Failure Trace



Roman Numerals



RomanNumeralsTestCase.java

```
    @Test
    public void testOne() {
        assertEquals("I", RomanNumerals.of(1));
    }

    @Test
    public void testTwo() {
        assertEquals("II", RomanNumerals.of(2));
    }

    @Test
    public void testThree() {
        assertEquals("III", RomanNumerals.of(3));
    }

    @Test
    public void testTen() {
        assertEquals("X", RomanNumerals.of(10));
    }

    @Test
    public void testTwenty() {
        assertEquals("XX", RomanNumerals.of(20));
    }
}
```

RomanNumerals.java

```
package tdd.romannumerals;

public class RomanNumerals {

    public static String of(int value) {
        String romanValue = "";

        // extract the pattern

        while (value >= 10) {
            romanValue += "X";
            value -= 10;
        }

        while (value >= 1) {
            romanValue += "I";
            value -= 1;
        }
```

JUnit

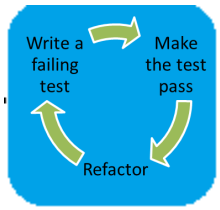
Finished after 0.024 seconds

Runs: 5/5 Errors: 0 Failures: 0

tdd.romannumerals.RomanNumeralsTestCase [Runner: JUnit 4] (0.

Failure Trace

Roman Numerals



RomanNumeralsTestCase.java

```
    @Test
    public void testOne() {
        assertEquals("I", RomanNumerals.of(1));
    }

    @Test
    public void testTwo() {
        assertEquals("II", RomanNumerals.of(2));
    }

    @Test
    public void testThree() {
        assertEquals("III", RomanNumerals.of(3));
    }

    @Test
    public void testTen() {
        assertEquals("X", RomanNumerals.of(10));
    }

    @Test
    public void testTwenty() {
        assertEquals("XX", RomanNumerals.of(20));
    }
}
```

RomanNumerals.java

```
public class RomanNumerals {

    public static String of(int value) {
        String romanValue = "";

        int[] arabic = new int[] {10, 1};
        String[] roman = new String[] {"X", "I"};

        for (int i = 0; i < 2; i ++) {
            while (value >= arabic[i]) {
                romanValue += roman[i];
                value -= arabic[i];
            }
        }

        return romanValue;
    }
```

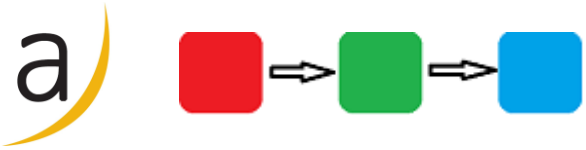
JUnit

Finished after 0.031 seconds

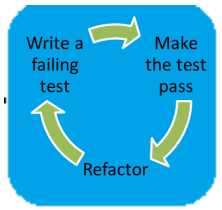
Runs: 5/5 Errors: 0 Failures: 0

tdd.romannumerals.RomanNumeralsTestCase [Runner: JUnit 4] (0.0

Failure Trace



Roman Numerals



RomanNumeralsTestCase.java

```
    @Test
    public void testOne() {
        assertEquals("I", RomanNumerals.of(1));
    }

    @Test
    public void testTwo() {
        assertEquals("II", RomanNumerals.of(2));
    }

    @Test
    public void testThree() {
        assertEquals("III", RomanNumerals.of(3));
    }

    @Test
    public void testTen() {
        assertEquals("X", RomanNumerals.of(10));
    }

    @Test
    public void testTwenty() {
        assertEquals("XX", RomanNumerals.of(20));
    }
}
```

RomanNumerals.java

```
public class RomanNumerals {
    // these seem to be better defined as "static" definitions
    private static final int[] arabic = new int[] {10, 1};
    private static final String[] roman = new String[] {"X", "I"};

    public static String of(int value) {
        String romanValue = "";

        for (int i = 0; i < 2; i ++) {
            while (value >= arabic[i]) {
                romanValue += roman[i];
                value -= arabic[i];
            }
        }

        return romanValue;
    }
```

JUnit

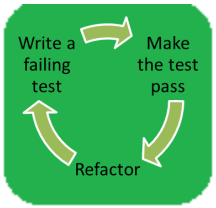
Finished after 0.024 seconds

Runs: 5/5 Errors: 0 Failures: 0

tdd.romannumerals.RomanNumeralsTestCase [Runner: JUnit 4] (0.

Failure Trace

Roman Numerals



RomanNumeralsTestCase.java

```
    @Test
    public void testTwo() {
        assertEquals("II", RomanNumerals.of(2));
    }

    @Test
    public void testThree() {
        assertEquals("III", RomanNumerals.of(3));
    }

    @Test
    public void testTen() {
        assertEquals("X", RomanNumerals.of(10));
    }

    @Test
    public void testTwenty() {
        assertEquals("XX", RomanNumerals.of(20));
    }

    @Test // the 30 test will pass, what about 33 ?
    public void testThrityThree() {
        assertEquals("XXXIII", RomanNumerals.of(33));
    }
}
```

RomanNumerals.java

```
public class RomanNumerals {
    // these seem to be better defined as "static" definitions
    private static final int[] arabic = new int[] {10, 1};
    private static final String[] roman = new String[] {"X", "I"};

    public static String of(int value) {
        String romanValue = "";

        for (int i = 0; i < 2; i ++) {
            while (value >= arabic[i]) {
                romanValue += roman[i];
                value -= arabic[i];
            }
        }

        return romanValue;
    }
```

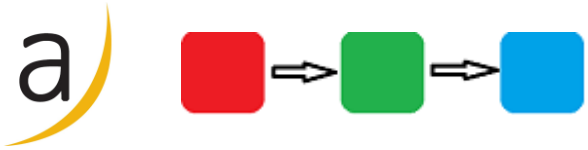
JUnit

Finished after 0.026 seconds

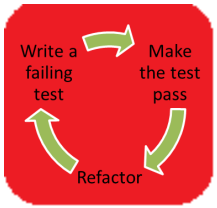
Runs: 6/6 Errors: 0 Failures: 0

tdd.romannumerals.RomanNumeralsTestCase [Runner: JUnit 4] (0.

Failure Trace



Roman Numerals



RomanNumeralsTestCase.java

```
18
19      @Test
20      public void testThree() {
21          assertEquals("III", RomanNumerals.of(3));
22      }
23
24      @Test
25      public void testTen() {
26          assertEquals("X", RomanNumerals.of(10));
27      }
28
29      @Test
30      public void testTwenty() {
31          assertEquals("XX", RomanNumerals.of(20));
32      }
33
34      @Test
35      public void testThrityThree() {
36          assertEquals("XXXIII", RomanNumerals.of(33));
37      }
38
39      @Test // I, X, ... C, M will probably have the same flow
40      public void testFive() { // 5 might be the first unclear test
41          assertEquals("V", RomanNumerals.of(5));
42      }
43  }
44
```

RomanNumerals.java

```
2
3
4   public class RomanNumerals {
5       // these seem to be better defined as "static" definitions
6       private static final int[] arabic = new int[] {10, 1};
7       private static final String[] roman = new String[] {"X", "I"};
8
9       public static String of(int value) {
10          String romanValue = "";
11
12          for (int i = 0; i < 2; i ++) {
13              while (value >= arabic[i]) {
14                  romanValue += roman[i];
15                  value -= arabic[i];
16              }
17          }
18
19          return romanValue;
20      }
```

JUnit

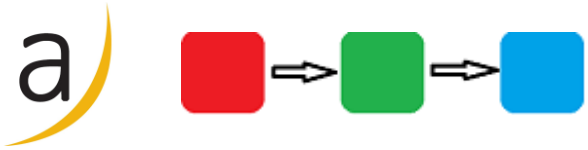
Finished after 0.032 seconds

Runs: 7/7 Errors: 0 Failures: 1

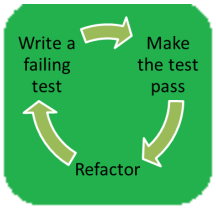
testTwo (0.000 s)
testThree (0.000 s)
testFive (0.007 s)
testThrityThree (0.000 s)

Failure Trace

org.junit.ComparisonFailure: expected:<[V]> but was:<[IIIII]>



Roman Numerals



RomanNumeralsTestCase.java

```
18
19    @Test
20    public void testThree() {
21        assertEquals("III", RomanNumerals.of(3));
22    }
23
24    @Test
25    public void testTen() {
26        assertEquals("X", RomanNumerals.of(10));
27    }
28
29    @Test
30    public void testTwenty() {
31        assertEquals("XX", RomanNumerals.of(20));
32    }
33
34    @Test
35    public void testThrityThree() {
36        assertEquals("XXXIII", RomanNumerals.of(33));
37    }
38
39    @Test // I, X, ... C, M will probably have the same flow
40    public void testFive() { // 5 might be the first unclear test
41        assertEquals("V", RomanNumerals.of(5));
42    }
43 }
44
```

RomanNumerals.java

```
2
3
4  public class RomanNumerals {
5      private static final int[] arabic = new int[] {10, 5, 1};
6      private static final String[] roman = new String[] {"X", "V", "I"};
7
8      public static String of(int value) {
9          String romanValue = "";
10
11         for (int i = 0; i < arabic.length; i ++) {
12             while (value >= arabic[i]) {
13                 romanValue += roman[i];
14                 value -= arabic[i];
15             }
16         }
17
18         return romanValue;
19     }
20
```

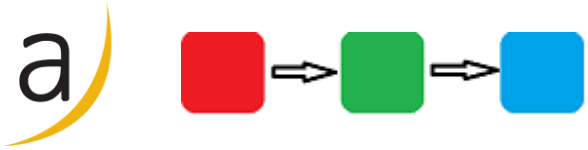
JUnit

Finished after 0.023 seconds

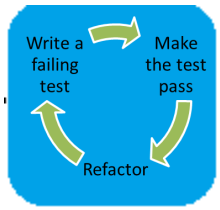
Runs: 7/7 Errors: 0 Failures: 0

tdd.romannumerals.RomanNumeralsTestCase [Runner: JUnit 4] (0.0

Failure Trace



Roman Numerals



RomanNumeralsTestCase.java

```
13
14      @Test
15      public void testTwo() {
16          assertEquals("II", RomanNumerals.of(2));
17      }
18
19      @Test
20      public void testThree() {
21          assertEquals("III", RomanNumerals.of(3));
22      }
23
24      @Test
25      public void testTen() {
26          assertEquals("X", RomanNumerals.of(10));
27      }
28
29      @Test
30      public void testTwenty() {
31          assertEquals("XX", RomanNumerals.of(20));
32      }
33
34      @Test
35      public void testThrityThree() {
36          assertEquals("XXXIII", RomanNumerals.of(33));
37      }
38
39      @Test // I, X, ... C, M will probably have the same flow
40      public void testFive() { // 5 might be the first unclear test
41          assertEquals("V", RomanNumerals.of(5));
42      }
43  }
44
```

RomanNumerals.java

```
 8  public class RomanNumerals {
 9      // it seems cleaner to have an ORDERED map
10      // instead of 2 arrays that work in "pairs"
11      private static final Map<Integer, String> pairs =
12              new LinkedHashMap<Integer, String>();
13      static {
14          pairs.put(10, "X");
15          pairs.put(5, "V");
16          pairs.put(1, "I");
17      };
18
19      public static String of(int value) {
20          String romanValue = "";
21
22          for (Entry<Integer, String> pair : pairs.entrySet())
23              while (value >= pair.getKey()) {
24                  romanValue += pair.getValue();
25                  value -= pair.getKey();
26              }
27
28          return romanValue;
29      }
30
```

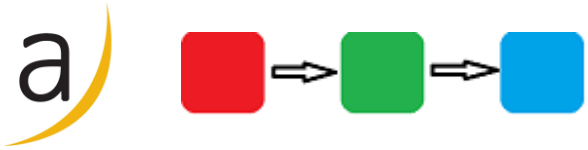
JUnit

Finished after 0.02 seconds

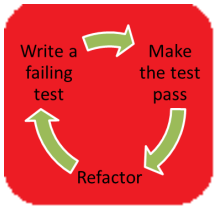
Runs: 7/7 Errors: 0 Failures: 0

tdd.romannumerals.RomanNumeralsTestCase [Runner: JUnit 4] (0.0

Failure Trace



Roman Numerals



RomanNumeralsTestCase.java

```
18
19      @Test
20      public void testThree() {
21          assertEquals("III", RomanNumerals.of(3));
22      }
23
24      @Test
25      public void testTen() {
26          assertEquals("X", RomanNumerals.of(10));
27      }
28
29      @Test
30      public void testTwenty() {
31          assertEquals("XX", RomanNumerals.of(20));
32      }
33
34      @Test
35      public void testThrityThree() {
36          assertEquals("XXXIII", RomanNumerals.of(33));
37      }
38
39      @Test
40      public void testFive() {
41          assertEquals("V", RomanNumerals.of(5));
42      }
43
44      @Test // 6, 16, 38, all seem to work - how about 4 ?
45      public void testFour() {
46          assertEquals("IV", RomanNumerals.of(4));
47      }
48  }
49
```

RomanNumerals.java

```
 8  public class RomanNumerals {
 9      // it seems cleaner to have an ORDERED map
10      // instead of 2 arrays that work in "pairs"
11      private static final Map<Integer, String> pairs =
12              new LinkedHashMap<Integer, String>();
13      static {
14          pairs.put(10, "X");
15          pairs.put(5, "V");
16          pairs.put(1, "I");
17      };
18
19      public static String of(int value) {
20          String romanValue = "";
21
22          for (Entry<Integer, String> pair : pairs.entrySet())
23              while (value >= pair.getKey()) {
24                  romanValue += pair.getValue();
25                  value -= pair.getKey();
26              }
27
28          return romanValue;
29      }
30
```

JUnit

Finished after 0.032 seconds

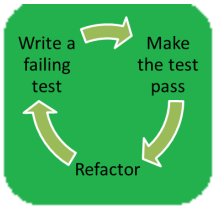
Runs: 8/8 Errors: 0 Failures: 1

testTen (0.000 s)
testTwo (0.000 s)
testThree (0.000 s)
testFive (0.000 s)
testFour (0.009 s)

Failure Trace

org.junit.ComparisonFailure: expected:<I[V]> but was:<I[II]>
at tdd.romannumerals.RomanNumeralsTestCase.testFour(RomanNum

Roman Numerals



RomanNumeralsTestCase.java

```
18
19      @Test
20      public void testThree() {
21          assertEquals("III", RomanNumerals.of(3));
22      }
23
24      @Test
25      public void testTen() {
26          assertEquals("X", RomanNumerals.of(10));
27      }
28
29      @Test
30      public void testTwenty() {
31          assertEquals("XX", RomanNumerals.of(20));
32      }
33
34      @Test
35      public void testThrityThree() {
36          assertEquals("XXXIII", RomanNumerals.of(33));
37      }
38
39      @Test
40      public void testFive() {
41          assertEquals("V", RomanNumerals.of(5));
42      }
43
44      @Test // 6, 16, 38, all seem to work - how about 4 ?
45      public void testFour() {
46          assertEquals("IV", RomanNumerals.of(4));
47      }
48  }
49
```

RomanNumerals.java

```
8   public class RomanNumerals {
9       // it seems cleaner to have an ORDERED map
10      // instead of 2 arrays that work in "pairs"
11      private static final Map<Integer, String> pairs =
12              new LinkedHashMap<Integer, String>();
13      static {
14          pairs.put(10, "X");
15          pairs.put(5, "V");
16          pairs.put(4, "IV");
17          pairs.put(1, "I");
18      };
19
20      public static String of(int value) {
21          String romanValue = "";
22
23          for (Entry<Integer, String> pair : pairs.entrySet())
24              while (value >= pair.getKey()) {
25                  romanValue += pair.getValue();
26                  value -= pair.getKey();
27              }
28
29          return romanValue;
30      }
```

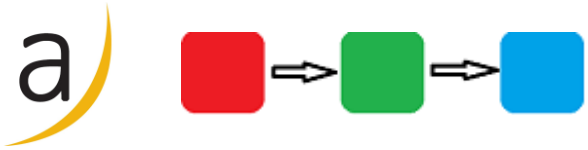
JUnit

Finished after 0.029 seconds

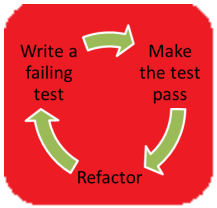
Runs: 8/8 Errors: 0 Failures: 0

tdd.romannumerals.RomanNumeralsTestCase [Runner: JUnit 4] (0.0

Failure Trace



Roman Numerals



RomanNumeralsTestCase.java

```
23
24      @Test
25      public void testTen() {
26          assertEquals("X", RomanNumerals.of(10));
27      }
28
29      @Test
30      public void testTwenty() {
31          assertEquals("XX", RomanNumerals.of(20));
32      }
33
34      @Test
35      public void testThrityThree() {
36          assertEquals("XXXIII", RomanNumerals.of(33));
37      }
38
39      @Test
40      public void testFive() {
41          assertEquals("V", RomanNumerals.of(5));
42      }
43
44      @Test
45      public void testFour() {
46          assertEquals("IV", RomanNumerals.of(4));
47      }
48
49      @Test
50      public void testNine() {
51          assertEquals("IX", RomanNumerals.of(9));
52      }
53  }
54
```

RomanNumerals.java

```
8   public class RomanNumerals {
9       private static final Map<Integer, String> pairs = new LinkedHashMap<
10      static {
11          pairs.put(10, "X");
12          pairs.put(5, "V");
13          pairs.put(4, "IV");
14          pairs.put(1, "I");
15      };
16
17      public static String of(int value) {
18          String romanValue = "";
19
20          for (Entry<Integer, String> pair : pairs.entrySet())
21              while (value >= pair.getKey()) {
22                  romanValue += pair.getValue();
23                  value -= pair.getKey();
24              }
25
26          return romanValue;
27      }
28
29  }
30
```

JUnit

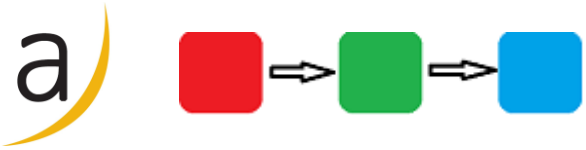
Finished after 0.037 seconds

Runs: 9/9 Errors: 0 Failures: 1

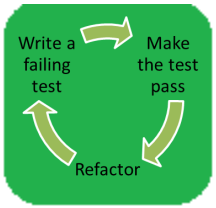
testTwo (0.000 s)
testThree (0.000 s)
testFive (0.000 s)
testFour (0.000 s)
testNine (0.000 s)

Failure Trace

org.junit.ComparisonFailure: expected:<[IX]> but was:<[VIV]>
at tdd.romannumerals.RomanNumeralsTestCase.testNine(RomanNum



Roman Numerals



RomanNumeralsTestCase.java

```
23
24      @Test
25      public void testTen() {
26          assertEquals("X", RomanNumerals.of(10));
27      }
28
29      @Test
30      public void testTwenty() {
31          assertEquals("XX", RomanNumerals.of(20));
32      }
33
34      @Test
35      public void testThrityThree() {
36          assertEquals("XXXIII", RomanNumerals.of(33));
37      }
38
39      @Test
40      public void testFive() {
41          assertEquals("V", RomanNumerals.of(5));
42      }
43
44      @Test
45      public void testFour() {
46          assertEquals("IV", RomanNumerals.of(4));
47      }
48
49      @Test
50      public void testNine() {
51          assertEquals("IX", RomanNumerals.of(9));
52      }
53  }
54
```

RomanNumerals.java

```
 8  public class RomanNumerals {
 9      private static final Map<Integer, String> pairs = new LinkedHashMap<
10      static {
11          pairs.put(10, "X");
12          pairs.put(9, "IX");
13          pairs.put(5, "V");
14          pairs.put(4, "IV");
15          pairs.put(1, "I");
16      };
17
18      public static String of(int value) {
19          String romanValue = "";
20
21          for (Entry<Integer, String> pair : pairs.entrySet())
22              while (value >= pair.getKey()) {
23                  romanValue += pair.getValue();
24                  value -= pair.getKey();
25              }
26
27          return romanValue;
28      }
29
30  }
```

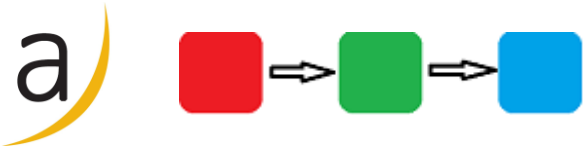
JUnit

Finished after 0.021 seconds

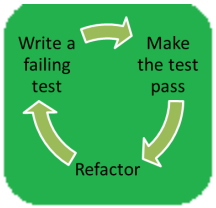
Runs: 9/9 Errors: 0 Failures: 0

tdd.romannumerals.RomanNumeralsTestCase [Runner: JUnit 4] (0.0

Failure Trace



Roman Numerals



RomanNumeralsTestCase.java

```
28
29      @Test
30      public void testTwenty() {
31          assertEquals("XX", RomanNumerals.of(20));
32      }
33
34      @Test
35      public void testThrityThree() {
36          assertEquals("XXXIII", RomanNumerals.of(33));
37      }
38
39      @Test
40      public void testFive() {
41          assertEquals("V", RomanNumerals.of(5));
42      }
43
44      @Test
45      public void testFour() {
46          assertEquals("IV", RomanNumerals.of(4));
47      }
48
49      @Test
50      public void testNine() {
51          assertEquals("IX", RomanNumerals.of(9));
52      }
53
54      @Test // the final implementation is "obvious"
55      public void test2699() {
56          assertEquals("MMDCXCIX", RomanNumerals.of(2699));
57      }
58  }
59
```

RomanNumerals.java

```
 8  public class RomanNumerals {
 9      private static final Map<Integer, String> pairs = new LinkedHashMap<
10      static {
11          pairs.put(1000, "M");
12          pairs.put(900, "CM");
13          pairs.put(500, "D");
14          pairs.put(400, "CD");
15          pairs.put(100, "C");
16          pairs.put(90, "XC");
17          pairs.put(50, "L");
18          pairs.put(40, "XL");
19          pairs.put(10, "X");
20          pairs.put(9, "IX");
21          pairs.put(5, "V");
22          pairs.put(4, "IV");
23          pairs.put(1, "I");
24      };
25
26      public static String of(int value) {
27          String romanValue = "";
28
29          for (Entry<Integer, String> pair : pairs.entrySet())
30              while (value >= pair.getKey()) {
```

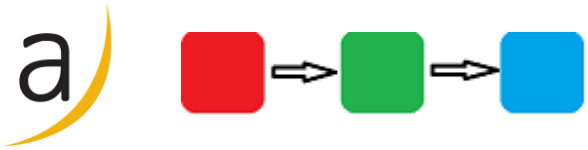
JUnit

Finished after 0.025 seconds

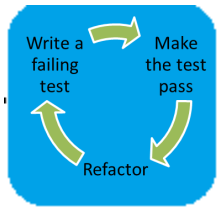
Runs: 10/10 Errors: 0 Failures: 0

tdd.romannumerals.RomanNumeralsTestCase [Runner: JUnit 4] (0.

Failure Trace



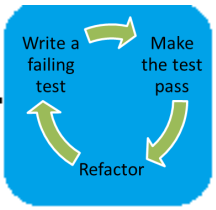
Roman Numerals



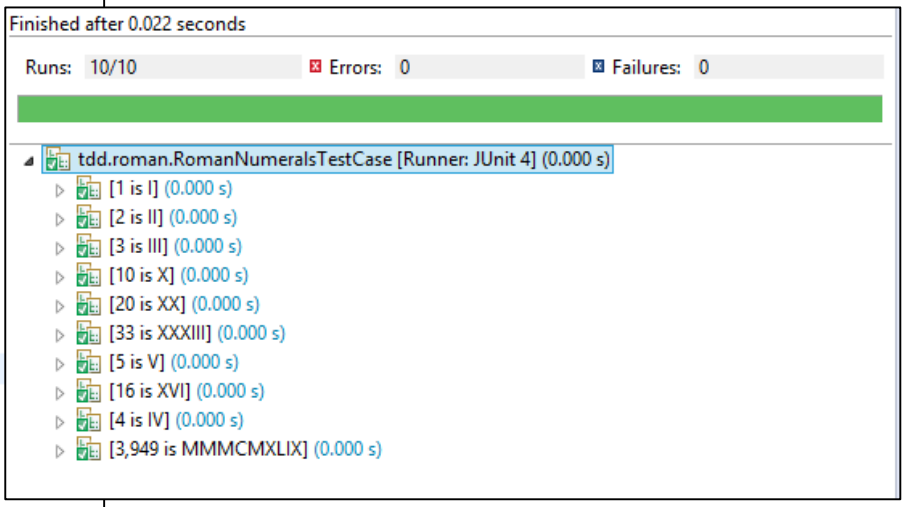

```
private static final Map<Integer, String> pairs = new LinkedHashMap<Integer, String>();
static {
    pairs.put(1000, "M");
    pairs.put(900, "CM");
    pairs.put(500, "D");
    pairs.put(400, "CD");
    pairs.put(100, "C");
    pairs.put(90, "XC");
    pairs.put(50, "L");
    pairs.put(40, "XL");
    pairs.put(10, "X");
    pairs.put(9, "IX");
    pairs.put(5, "V");
    pairs.put(4, "IV");
    pairs.put(1, "I");
};
```

```
public static String of(int value) {
    String romanValue = "";

    for (Entry<Integer, String> pair : pairs.entrySet())
        while (value >= pair.getKey()) {
            romanValue += pair.getValue();
            value -= pair.getKey();
        }

    return romanValue;
}
```

Roman Numerals




```java
@RunWith(Parameterized.class)
public class RomanNumeralsTestCase {
    private final int arabic;
    private final String roman;

    public RomanNumeralsTestCase(int arabic, String roman) {
        this.arabic = arabic;
        this.roman = roman;
    }

    @Parameters(name="{0} is {1}")
    public static List<Object[]> parametersFeed() {
        List<Object[]> feed = new ArrayList<Object[]>();

        feed.add(new Object[] {1, "I"});
        feed.add(new Object[] {2, "II"});
        feed.add(new Object[] {3, "III"});
        feed.add(new Object[] {10, "X"});
        feed.add(new Object[] {20, "XX"});
        feed.add(new Object[] {33, "XXXIII"});
        feed.add(new Object[] {5, "V"});
        feed.add(new Object[] {16, "XVI"});
        feed.add(new Object[] {4, "IV"});
        feed.add(new Object[] {3949, "MMMCMXLIX"});

        return feed;
    }

    @Test
    public void transform() {
        assertEquals(roman, get(arabic));
    }
}
```



Array Sort

Array Sort

8	7	9	21	5	9	5
5	5	7	8	9	9	21

Comparison sorts

Name	Best	Average	Worst	Memory	Stabl
Quicksort	$n \log n$ variation is n	$n \log n$	n^2	$\log n$ on average, worst case is n; Sedgewick variation is $\log n$ worst case	typical in-plac stable; stable
Merge sort	$n \log n$	$n \log n$	$n \log n$	n	Ye
In-place merge sort	—	—	$n \log^2 n$	1	Ye
Heapsort	$n \log n$	$n \log n$	$n \log n$	1	No
Insertion sort	n	n^2	n^2	1	Ye
Introsort	$n \log n$	$n \log n$	$n \log n$	$\log n$	No
Selection sort	n^2	n^2	n^2	1	No
Timsort	n	$n \log n$	$n \log n$	n	Ye

Array Sort

```java
import static tdd.bowlingscore.SortUtil.sort;

import org.junit.Test;

public class SortTestCase {

    @Test
    public void sortNullArray() {
        assertArrayEquals(null, sort(null));
    }

    @Test
    public void sortOneElement() {
        assertArrayEquals(new int[] {1},
                sort(new int[] {1}));
    }

    @Test
    public void sortTwoInOrder() {
        assertArrayEquals(new int[] {1, 2},
                sort(new int[] {1, 2}));
    }

    @Test
    public void sortTwoReversed() {
        assertArrayEquals(new int[] {1, 2},
                sort(new int[] {1, 2}));
    }

    @Test
    public void sortThreeInOrder() {
        assertArrayEquals(new int[] {1, 2, 3},
                sort(new int[] {1, 2, 3}));
    }

    @Test
    public void sortThreeLastTwoReversed() {
        assertArrayEquals(new int[] {1, 2, 3},
                sort(new int[] {1, 3, 2}));
    }

    @Test
    public void sortThreeFirstTwoReversed() {
        assertArrayEquals(new int[] {1, 2, 3},
                sort(new int[] {2, 1, 3}));
    }

    @Test
    public void sortThreeReversed() {
        assertArrayEquals(new int[] {1, 2, 3},
                sort(new int[] {3, 2, 1}));
    }

    @Test
    public void sortAllTheSame() {
        assertArrayEquals(new int[] {1, 1, 1},
                sort(new int[] {1, 1, 1}));
    }

    @Test
    public void sortComplexArray() {
        assertArrayEquals(new int[] {1, 1, 2, 3, 4, 5, 5, 5, 8, 9, 10},
                sort(new int[] {5, 10, 1, 3, 2, 1, 4, 5, 5, 8, 9}));
    }
}
```

Array Sort

```
3  public class SortUtil {
4
5      public static int[] sort(int[] elements) {
6          if (elements == null) {
7              return null;
8          }
9
10         boolean changed = true;
11         while (changed) {
12             changed = false;
13             for (int i = 0; i < elements.length - 1; i ++) {
14                 if (elements[i] > elements[i+1]) {
15                     int aux = elements[i+1];
16                     elements[i+1] = elements[i];
17                     elements[i] = aux;
18
19                     changed = true;
20                 }
21             }
22         }
23
24         return elements;
25     }
26
```

Fizz Buzz (Bolț)

Fizz Buzz (Bolț)

Fizz buzz

From Wikipedia, the free encyclopedia

Fizz buzz is a group word game for children to teach them about division.[1] Players take turns to count incrementally, replacing any number divisible by three with the word "fizz", and any number divisible by five with the word "buzz".

1, 2, Fizz, 4, Buzz, Fizz, 7, 8, Fizz, Buzz, 11, Fizz, 13, 14, Fizz Buzz, 16, 17, Fizz, 19, Buzz, Fizz, 22, 23, Fizz, Buzz, 26, Fizz, 28, 29, Fizz Buzz, 31, 32, Fizz, 34, Buzz, Fizz, ...

Fizz Buzz (Bolț)

```
 9  public class FizzBuzzTestCase {
10
11      @Test
12      public void testOne() {
13          assertEquals("1", say(1));
14      }
15
16      @Test
17      public void testTwo() {
18          assertEquals("2", say(2));
19      }
20
21      @Test
22      public void testThree() {
23          assertEquals("Fizz", say(3));
24      }
25
26      @Test
27      public void testFive() {
28          assertEquals("Buzz", say(5));
29      }
```

```
31      @Test
32      public void testSix() {
33          assertEquals("Fizz", say(6));
34      }
35
36      @Test
37      public void testTen() {
38          assertEquals("Buzz", say(10));
39      }
40
41      @Test
42      public void testFiteen() {
43          assertEquals("FizzBuzz", say(15));
44      }
45  }
```

Fizz Buzz (Bolț)

```
3  public class FizzBuzz {
4
5      public static String say(int n) {
6          String sayWhat = String.valueOf(n);
7          if (divisible(n, 3))
8              sayWhat = "Fizz";
9
10         if (divisible(n, 5))
11             sayWhat = "Buzz";
12
13         if (divisible(n, 15))
14             sayWhat = "FizzBuzz";
15
16         return sayWhat;
17     }
18
19     private static boolean divisible(int n, int divisor) {
20         return n % divisor == 0;
21     }
22 }
```

TDD – Why ?

1. Tests are the documentation – what does this function do ? How is that supposed to work ?

2. 10 minutes ago everything worked and was shippable

3. Small steps – avoid existing bugs from breaking your new code

4. The test is not there to only validate your code – use it as a design, use it for examples, use it for expectations

TDD – Why ?

5. Requires examples prior to implementation (we need what, not how)

6. QA = quality assurance – focus on performance, on UI, on UX, on features, on anything other than “does it work”

7. Would you accept anything less than a tested application ? Would you fly on a plane with software developed without any tests ?

8. It is not complete yes – but it is correct; At least to this point in time;

TDD – Why ?

8. How much debugging do you imagine doing ? (if it all worked 5 minutes ago).

9. You see bad, ugly, old, messy code. Someone needs to clean it – but not me; Because I might break it;

10. We want our code flexible, maintainable; A perfectly designed system with no tests, we will be afraid to change it and it will rot;

TDD – Why ?

11. If we had 90-95% coverage; When the tests all pass – we can ship it – we are confident it “works”

12. When (if) you write the tests at the end – you just write them because someone said you should. You already trust it works because you tested it manually - but – would you change and clean it a month later – with complete confidence ? How about 6 months later ?

TDD – Why not …

1. It’s harder
2. It takes longer
3. When you change something – you also have the tests to bother with
4. I don’t have all the answers (examples / requirements) yet
5. The project is to easy to require TDD

References

http://butunclebob.com

http://agiledata.org/essays/tdd.html

http://code.tutsplus.com/tutorials/the-newbies-guide-to-test-driven-development--net-13835

https://en.wikipedia.org/wiki/Sorting_algorithm

http://langrsoft.com/jeff/2011/09/tdd-kata-roman-number-converter/

http://c2.com/cgi/wiki?PrematureOptimization