Delivering Better Software Faster with Test-Driven Development

Burk Hufnagel - Solution Architect
Daugherty Business Solutions

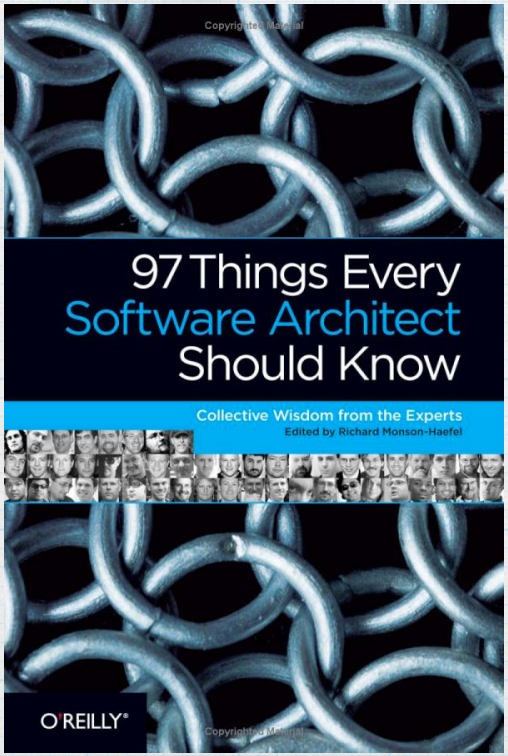
Who is Burk Hufnagel?

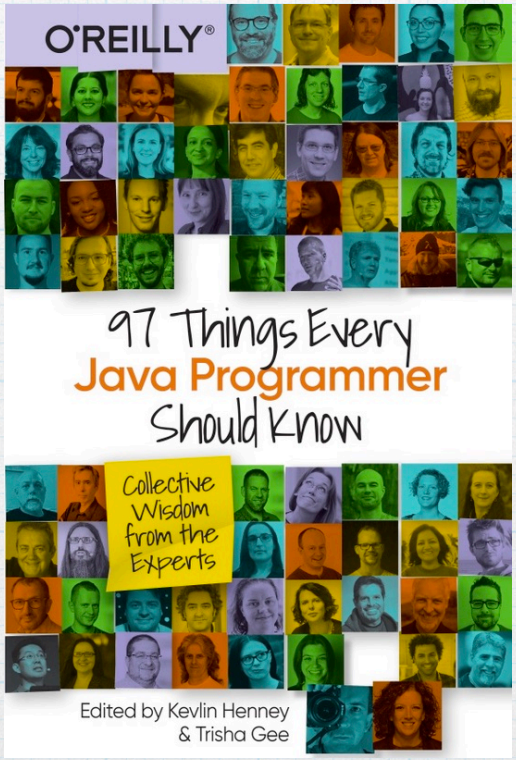
Programmer\Solution Architect with

Daugherty Business Solutions

Java™ Champions

JavaOne Rock Star





Goals

* Introducing TDD
* Succeeding with TDD
* Starting with TDD in the Real World
* Summary

Common Concerns

* TANSTAAFL

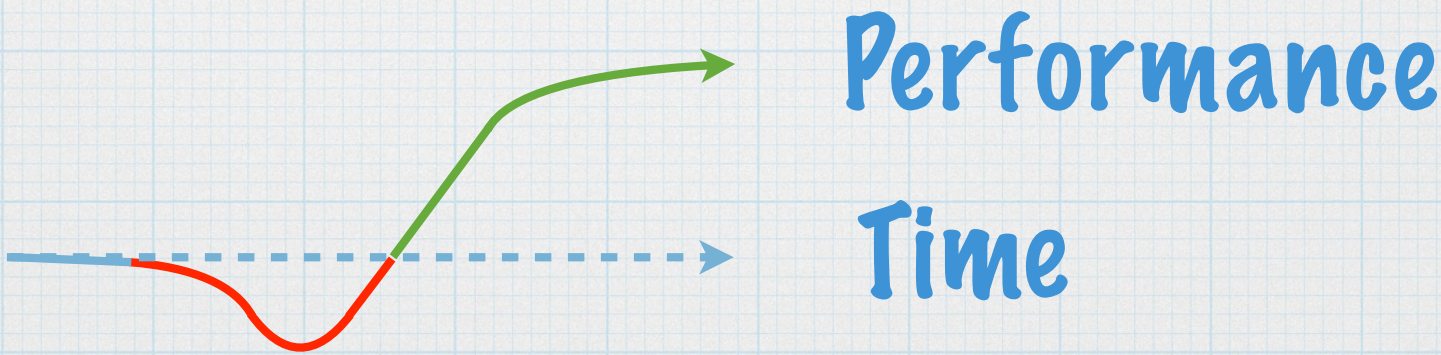
Common Concerns

* TANSTAAFL

TANSTAAFL

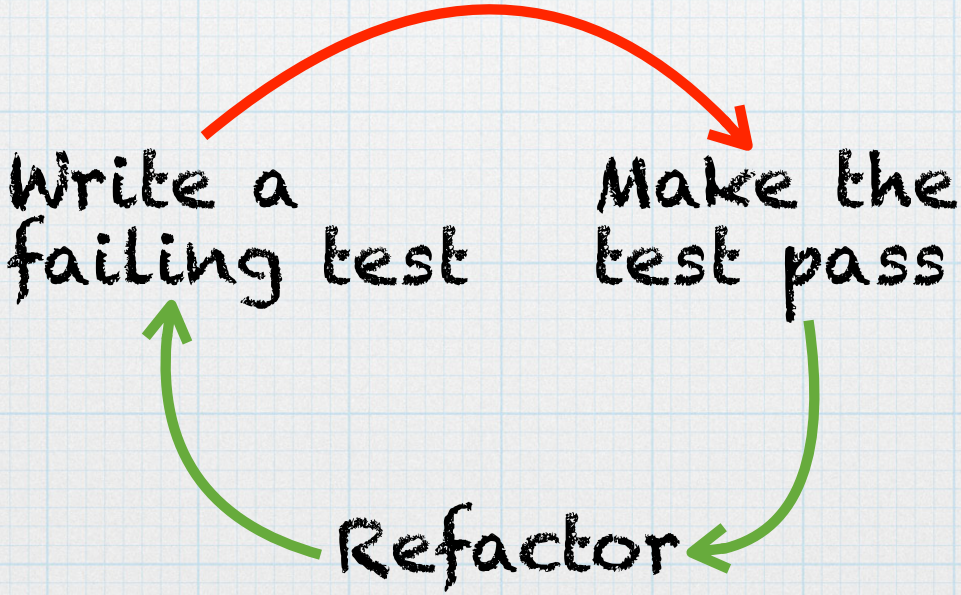
There Ain't No Such Thing As A Free Lunch

* TDD brings lots of benefits, but it also takes time to get good at it.
* Expect to see a dip in performance as you're learning, then an increase as you get used to it.
* It's natural and happens whenever you learn to do something new, or use a new tool



Test-Driven Development

* TDD is an efficient, three step process
  * Write a failing automated test
  * Write production code to make it pass
  * Refactor to clean up any mess



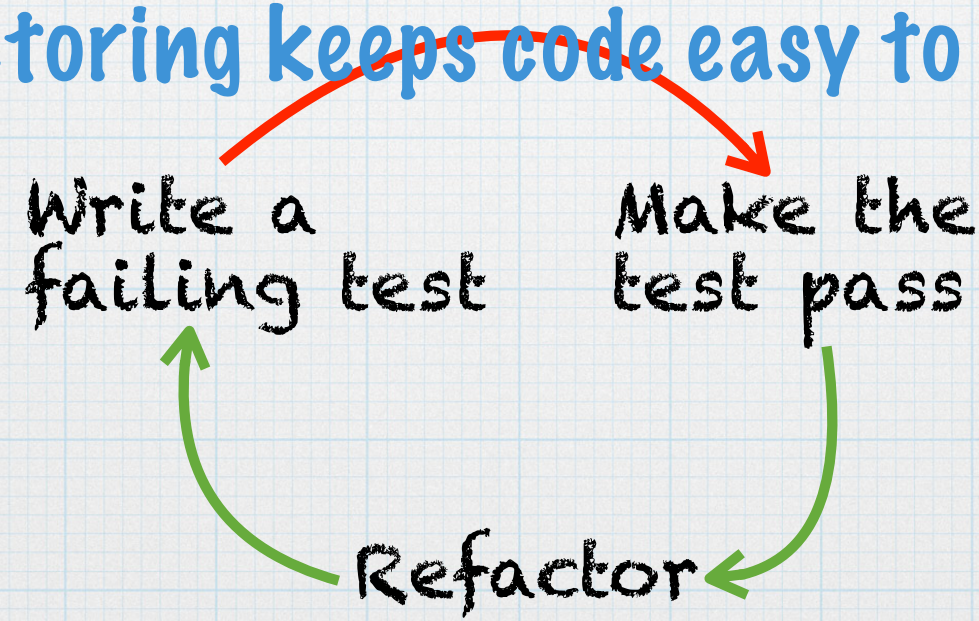
An automated test

```
class StackTest {

  @Test
  void callingPopShouldReturnTheLastValuePushed() {
      // Given 42 and 23 are pushed onto an empty Stack
      Stack stack = new Stack();
      stack.push(42);
      int expectedValue = 23;
      stack.push(expectedValue);

      // When pop() is called
      int actualValue = stack.pop();

      // Then it should return 23
      assertEquals(expectedValue, actualValue);
  }
}
```

Critical! →

What makes TDD efficient?

* Tests are based on requirements
* Only needed production code is written
* Decouples design from implementation
* Tests fail when unexpected changes occur
* Refactoring keeps code easy to maintain



Common Process vs TDD

Read requirements

Analysis

Write production code

Manually test code

Use debugger to find & fix bugs

Turn over to QA

Find bugs

Turn over to UAT

Find bugs

Release to production

Find bugs

Read requirements

Analysis

Write a failing test

Write code to pass test

Refactor

Turn over to QA

Find fewer bugs

Turn over to UAT

Find fewer bugs

Release to production

Find fewer bugs

Goals

* Introducing TDD
* Succeeding with TDD
* Doing TDD in the Real World
* Summary

Common Concerns

* TANSTAAFL
* Writing tests first is illogical. How can you test code that doesn't exist?

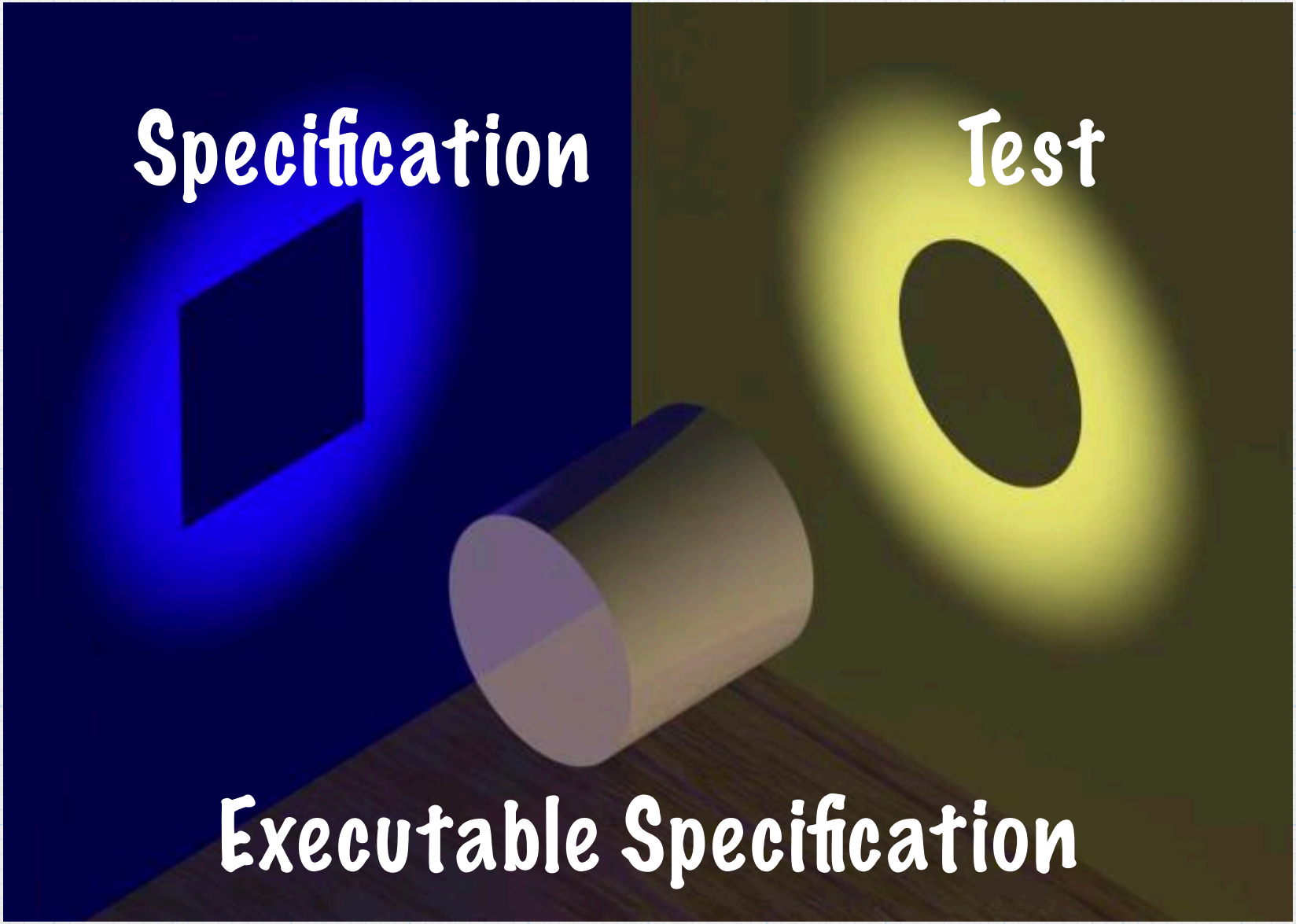
The real problem

* Testing is what you do after you do something else
* Specifications describe an expected behavior... but doesn't verify it
* Executable specifications do both!

Executable Specifications

* Based on requirements
  * Must understand the details. No guessing allowed; ask business folks
* Describe what should happen from the user's viewpoint. No implementation details!

Executable Specification

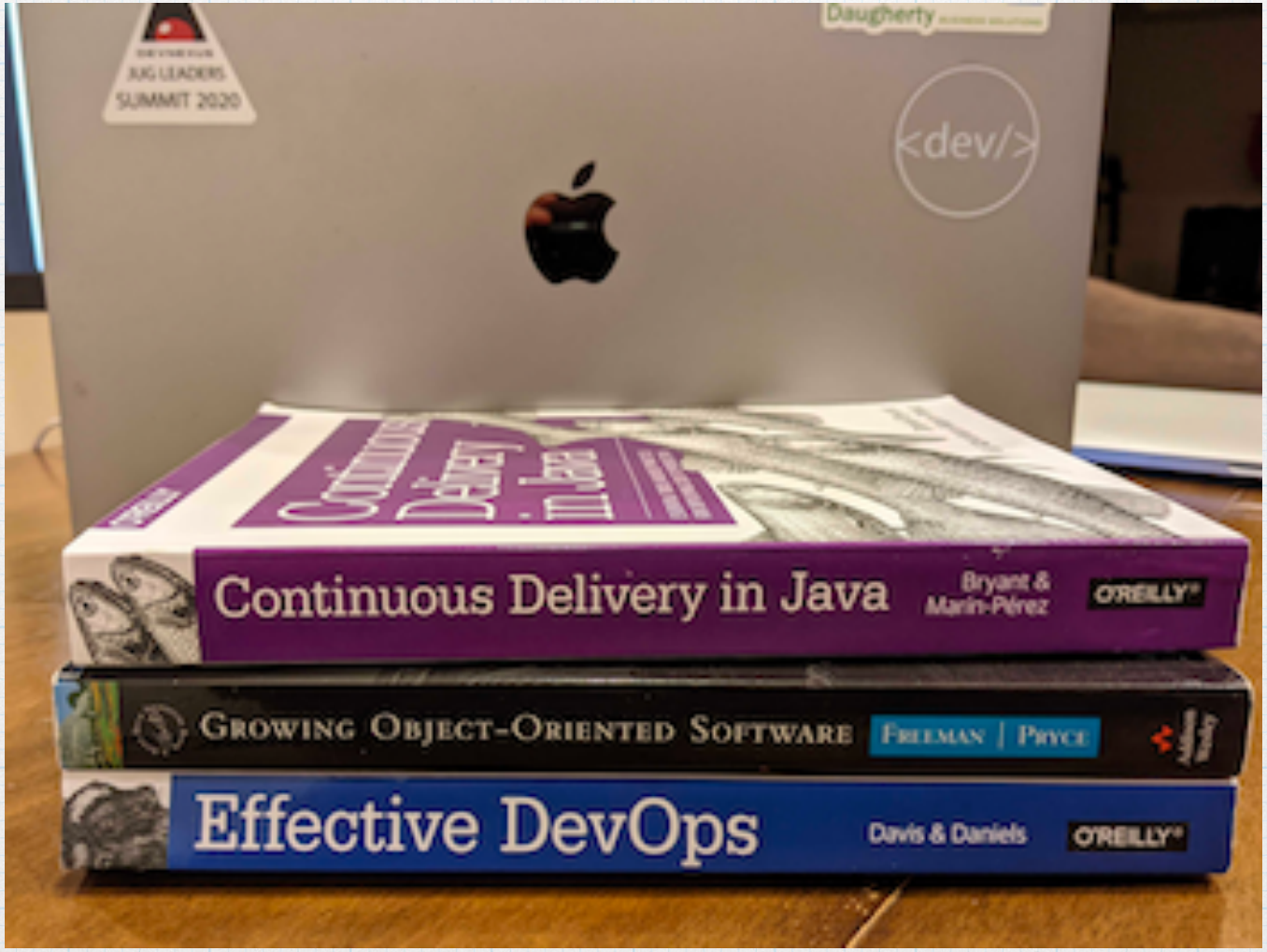


Common Concerns

* TANSTAAFL
* Writing tests first is illogical. How can you test code that doesn't exist?
* How do I know what tests to write?
* How do I know when I'm done?

Let's TDD a Stack

A stack stores and returns data in a Last-in-First-out manner; like a stack of books or plates on a table.



Stack Requirements - Biz

A stack stores and returns data in a Last-in-First-out manner; like a stack of books or plates on a table.

* push() — Adds an item to the top of the stack.
* pop() — Removes & returns top item. If it's empty, throws EmptyStackException.
* isEmpty() — Returns true when no items are stored and false when there are.
* peek() — Returns a reference to the top item. If it's empty, throws EmptyStackException.

Note: Prioritized by the business, based on expected value

Dev Team Analysis

A stack stores and returns data in a Last-in-First-out manner; like a stack of books or plates on a table.

* No limit to the number of items it can store is mentioned. Is this an error, or intentional?
* No external way to test push() without pop() or peek() already working
* Agree push() and pop() are critical, but believe peek() and isEmpty() can be added later

Stack Requirements - Dev

A stack stores and returns data in a Last-in-First-out manner; like a stack of books or plates on a table.

* pop() — Removes & returns top item. If it's empty, throws EmptyStackException
* push() — Adds an item to the top of the stack
* isEmpty() — Returns true when no items are stored and false when there are
* peek() — Returns a reference to the top item. If it's empty, throws EmptyStackException

Stack Scenarios

Given a new Stack:

* Calling pop should throw an EmptyStackException.
* Calling pop after pushing 5, should return 5. Calling pop again should throw an EmptyStackException.
* Pushing 2 and 3, then calling pop() twice should return 3, 2.

```
class StackSpec extends Specification {
    def "Calling pop() on a new Stack should throw an EmptyStackException"() {
        given: "a new instance"


        when: "pop() is called"


        then: "it should throw an EmptyStackException"


    }

}
```

Generated documentation

/Users/bth0624/git/tdd-a-stack/build/spock-reports/com.grokspock.dev...

Report for com.grokspock.devnexus2021.StackSpecification

<< Back

Summary:

Created on Sun Jan 31 22:04:11 EST 2021 by bth0624

Executed features	Passed	Failures	Errors	Skipped	Success rate	Time
1	1	0	0	0	100.0%	0.039 seconds

Features:

- Calling pop() on a new Stack should throw an EmptyStackException

Calling pop() on a new Stack should throw an EmptyStackException

Return

Given: a new instance
When: pop() is called
Then: it should throw an EmptyStackException

Generated by Athaydes Spock Reports

Link to spec

Method name

Spec text

Return link

```
class StackSpec extends Specification {
    def "Calling pop() on a new Stack should throw an EmptyStackException"() {
        given: "a new instance"


        when: "pop() is called"


        then: "it should throw an EmptyStackException"


    }

}
```

Uncle Bob's 3 Laws of TDD

1. Only write production code when there is a failing unit test.
2. Only write enough of a test for it to fail, and not compiling counts.
3. Only write enough production code to make the failing test pass.

```
class StackSpec extends Specification {
    def "Calling pop() on a new Stack should throw an EmptyStackException"() {
        given: "a new instance"
        def stack = new Stack()

        when: "pop() is called"


        then: "it should throw an EmptyStackException"


    }

}
```

```
class StackSpec extends Specification {
    def "Calling pop() on a new Stack should throw an EmptyStackException"() {
        given: "a new instance"
        def stack = new Stack()

        when: "pop() is called"


        then: "it should throw an EmptyStackException"


    }

}
```

```
  public class Stack {


  }
```

```
class StackSpec extends Specification {
    def "Calling pop() on a new Stack should throw an EmptyStackException"() {
        given: "a new instance"
        def stack = new Stack()

        when: "pop() is called"


        then: "it should throw an EmptyStackException"


    }

}
```

```
  public class Stack {


  }
```

```
class StackSpec extends Specification {
    def "Calling pop() on a new Stack should throw an EmptyStackException"() {
        given: "a new instance"
        def stack = new Stack()

        when: "pop() is called"
        stack.pop()

        then: "it should throw an EmptyStackException"


    }

}
```

```
public class Stack {


}
```

```
class StackSpec extends Specification {
    def "Calling pop() on a new Stack should throw an EmptyStackException"() {
        given: "a new instance"
        def stack = new Stack()

        when: "pop() is called"
        stack.pop()

        then: "it should throw an EmptyStackException"


    }

}
```

```
public class Stack {
    public void pop() {

    }

}
```

```
class StackSpec extends Specification {
    def "Calling pop() on a new Stack should throw an EmptyStackException"() {
        given: "a new instance"
        def stack = new Stack()

        when: "pop() is called"
        stack.pop()

        then: "it should throw an EmptyStackException"


    }

}
```

```
  public class Stack {
      public void pop() {

      }

  }
```

```
class StackSpec extends Specification {
    def "Calling pop() on a new Stack should throw an EmptyStackException"() {
        given: "a new instance"
        def stack = new Stack()

        when: "pop() is called"
        stack.pop()

        then: "it should throw an EmptyStackException"
        thrown EmptyStackException

    }

}
```

```
public class Stack {
    public void pop() {

    }

}
```

```groovy
class StackSpec extends Specification {
    def "Calling pop() on a new Stack should throw an EmptyStackException"() {
        given: "a new instance"
        def stack = new Stack()

        when: "pop() is called"
        stack.pop()

        then: "it should throw an EmptyStackException"
        thrown EmptyStackException

    }
```

Run: StackSpecification

Tests failed: 1 of 1 test – 25 ms

Test Results	25 ms
StackSpecification	25 ms
Calling pop() on a new Stack	25 ms

```
Expected exception of type 'java.util.EmptyStackException', but no exception was thrown

    at org.spockframework.lang.SpecInternals.checkExceptionThrown(SpecInternals.java:81)
    at org.spockframework.lang.SpecInternals.thrownImpl(SpecInternals.java:68)
    at com.grokspock.devnexus2021.StackSpecification.Calling pop() on a new Stack should throw an EmptyStackException(StackSpecification.groovy:39)
```

```groovy
    public void pop() {

    }

}
```

```
class StackSpec extends Specification {
    def "Calling pop() on a new Stack should throw an EmptyStackException"() {
        given: "a new instance"
        def stack = new Stack()

        when: "pop() is called"
        stack.pop()

        then: "it should throw an EmptyStackException"
        thrown EmptyStackException

    }

}
```

```
public class Stack {
    public void pop() {

    }

}
```

```
class StackSpec extends Specification {
    def "Calling pop() on a new Stack should throw an EmptyStackException"() {
        given: "a new instance"
        def stack = new Stack()

        when: "pop() is called"
        stack.pop()

        then: "it should throw an EmptyStackException"
        thrown EmptyStackException

    }

}
```

```
public class Stack {
    public void pop() {
        throw new EmptyStackException();
    }

}
```

```
class StackSpec extends Specification {
    def "Calling pop() on a new Stack should throw an EmptyStackException"() {
        given: "a new instance"
        def stack = new Stack()

        when: "pop() is called"
        stack.pop()

        then: "it should throw an EmptyStackException"
        thrown EmptyStackException
```

Run: ToDoListApplication | StackSpecification.Calling pop() on empty Stack should throw...

Tests passed: 1 of 1 test – 27 ms

Test Results 27 ms
com.grokspock.devnexus2021.S 27 ms
Calling pop() on empty Stack shou

```
BUILD SUCCESSFUL in 4s
5 actionable tasks: 2 executed, 3 up-to-date
9:53:19 PM: Task execution finished ':test --tests "com.grokspock.devnexus2021.StackSpecification.Calling pop() on empty Stack should throw StackUnderflowExcep
```

```
public class Stack {
    public void pop() {
        throw new EmptyStackException();
    }

}
```

```
class StackSpec extends Specification {
    def "Calling pop() on a new Stack should throw an EmptyStackException"() {
        given: "a new instance"
        def stack = new Stack()

        when: "pop() is called"
        stack.pop()

        then: "it should throw an EmptyStackException"
        thrown EmptyStackException

    }
}
```

Refactor!

```
public class Stack {
    public void pop() {
        throw new EmptyStackException();
    }

}
```

Stack Scenarios

Given a new Stack:

* Calling pop should throw an EmptyStackException.
* Calling pop after pushing 5, should return 5. Calling pop again should throw an EmptyStackException.
* Pushing 2 and 3, then calling pop() twice should return 3, 2.

```
class StackSpec extends Specification {
    def "Calling push(5), pop() should return 5. Calling pop() again
         should throw an EmptyStackException"() {
        given: "we call push(5) on a new instance"


        expect: "when pop() is called, it should return 5"

        when: "pop() is called again"

        then: "it should throw an EmptyStackException"


    }

}
public class Stack {
    public void pop() {
        throw new EmptyStackException();
    }

}
```

```
class StackSpec extends Specification {
    def "Calling push(5), pop() should return 5. Calling pop() again
         should throw an EmptyStackException"() {
        given: "we call push(5) on a new instance"
            def stack = new Stack()
            stack.push(5)
        expect: "when pop() is called, it should return 5"

        when: "pop() is called again"

        then: "it should throw an EmptyStackException"


    }

}
public class Stack {
    public void pop() {
        throw new EmptyStackException();
    }

}
```

```
class StackSpec extends Specification {
    def "Calling push(5), pop() should return 5. Calling pop() again
         should throw an EmptyStackException"() {
        given: "we call push(5) on a new instance"
            def stack = new Stack()
            stack.push(5)
        expect: "when pop() is called, it should return 5"

        when: "pop() is called again"

        then: "it should throw an EmptyStackException"


    }

}
public class Stack {
    public void push(int value) {

    }


    public void pop() {
        throw new EmptyStackException();
    }

}
```

```
class StackSpec extends Specification {
    def "Calling push(5), pop() should return 5. Calling pop() again
         should throw an EmptyStackException"() {
        given: "we call push(5) on a new instance"
            def stack = new Stack()
            stack.push(5)
        expect: "when pop() is called, it should return 5"

        when: "pop() is called again"

        then: "it should throw an EmptyStackException"


    }

}
public class Stack {
    public void push(int value) {

    }


    public void pop() {
        throw new EmptyStackException();
    }

}
```

```
class StackSpec extends Specification {
    def "Calling push(5), pop() should return 5. Calling pop() again
         should throw an EmptyStackException"() {
        given: "we call push(5) on a new instance"
            def stack = new Stack()
            stack.push(5)
        expect: "when pop() is called, it should return 5"
            stack.pop() == 5
        when: "pop() is called again"

        then: "it should throw an EmptyStackException"


    }

}
public class Stack {
    public void push(int value) {

    }


    public void pop() {
        throw new EmptyStackException();
    }

}
```

```
class StackSpec extends Specification {
    def "Calling push(5), pop() should return 5. Calling pop() again
         should throw an EmptyStackException"() {
        given: "we call push(5) on a new instance"
            def stack = new Stack()
            stack.push(5)
        expect: "when pop() is called, it should return 5"
            stack.pop() == 5
        when: "pop() is called again"
            stack.pop()
        then: "it should throw an EmptyStackException"


    }

}
public class Stack {
    public void push(int value) {

    }


    public void pop() {
        throw new EmptyStackException();
    }

}
```

```
class StackSpec extends Specification {
    def "Calling push(5), pop() should return 5. Calling pop() again
         should throw an EmptyStackException"() {
        given: "we call push(5) on a new instance"
            def stack = new Stack()
            stack.push(5)
        expect: "when pop() is called, it should return 5"
            stack.pop() == 5
        when: "pop() is called again"
            stack.pop()
        then: "it should throw an EmptyStackException"
            thrown EmptyStackException

    }

}
public class Stack {
    public void push(int value) {

    }


    public void pop() {
        throw new EmptyStackException();
    }

}
```

```
class StackSpec extends Specification {
    def "Calling push(5), pop() should return 5. Calling pop() again
         should throw an EmptyStackException"() {
        given: "we call push(5) on a new instance"
            def stack = new Stack()
            stack.push(5)
        expect: "when pop() is called, it should return 5"
            stack.pop() == 5
        when: "pop() is called again"
            stack.pop()
        then: "it should throw an EmptyStackException"
            thrown EmptyStackException

    }
```

Run: StackSpecification

Tests failed: 1, passed: 1 of 2 tests – 233 ms

- Test Results 233 ms
  - StackSpecification 233 ms
    - Calling pop() on a new Stack 25 ms
    - Calling push(5), pop() shou 208 ms

```
Condition failed with Exception:

stack.pop() == 5
|     |
|     java.util.EmptyStackException
|       at com.grokspock.devnexus2021.Stack.pop(Stack.java:20)
|       at com.grokspock.devnexus2021.StackSpecification.Calling push(5), pop() should return 5. Calling pop() again should throw an EmptyStackException(StackS
<com.grokspock.devnexus2021.Stack@69b2f8e5 values=[5]>
```

```
    }


    public void pop() {
        throw new EmptyStackException();
    }

}
```

File | /Users/bth0624/git/tdd-a-stack/build/spock-reports/com.grokspock.dev...

Report for com.grokspock.devnexus2021.StackSpecification

<< Back

Summary:

Created on Sun Jan 31 21:35:09 EST 2021 by bth0624

Executed features	Passed	Failures	Errors	Skipped	Success rate	Time
2	1	1	0	0	50.0%	0.270 seconds

Features:

- Calling pop() on a new Stack should throw an EmptyStackException
- Calling push(5), pop() should return 5. Calling pop() again should throw an EmptyStackException

Green == passed
Red == failed

Calling pop() on a new Stack should throw an EmptyStackException

Return

Given: a new instance
When: pop() is called
Then: it should throw an EmptyStackException

Calling push(5), pop() should return 5. Calling pop() again should throw an EmptyStackException

Return

Given: we call push(5) on a new instance
Expect: when pop() is called, it should return 5
When: when calling pop() again
Then: it should throw an EmptyStackException

The following problems occurred:

- Condition failed with Exception:

```
stack.pop() == 5
|      |
|      java.util.EmptyStackException
|
at com.grokspock.devnexus2021.Stack.pop(Stack.java:16)
|
at com.grokspock.devnexus2021.StackSpecification.Calling push(5), pop() should return 5.
Calling pop() again should throw an EmptyStackException(StackSpecification.groovy:28)
<com.grokspock.devnexus2021.Stack@466d49f0 value=[5]>
```

Line that failed
Expected value
Actual value
Stack trace

Generated by Athaydes Spock Reports

```
class StackSpec extends Specification {
    def "Calling push(5), pop() should return 5. Calling pop() again
         should throw an EmptyStackException"() {
        given: "we call push(5) on a new instance"
            def stack = new Stack()
            stack.push(5)
        expect: "when pop() is called, it should return 5"
            stack.pop() == 5
        when: "pop() is called again"
            stack.pop()
        then: "it should throw an EmptyStackException"
            thrown EmptyStackException

    }

}
public class Stack {
    public void push(int value) {

    }


    public void pop() {
        throw new EmptyStackException();
    }

}
```

```
class StackSpec extends Specification {
    def "Calling push(5), pop() should return 5. Calling pop() again
         should throw an EmptyStackException"() {
        given: "we call push(5) on a new instance"
            def stack = new Stack()
            stack.push(5)
        expect: "when pop() is called, it should return 5"
            stack.pop() == 5
        when: "pop() is called again"
            stack.pop()
        then: "it should throw an EmptyStackException"
            thrown EmptyStackException

    }

}
public class Stack {
    public void push(int value) {
        this.value = value;
    }


    public int pop() {
        return value;
    }

}
```

```
class StackSpec extends Specification {
    def "Calling push(5), pop() should return 5. Calling pop() again
         should throw an EmptyStackException"() {
        given: "we call push(5) on a new instance"
            def stack = new Stack()
            stack.push(5)
        expect: "when pop() is called, it should return 5"
            stack.pop() == 5
        when: "pop() is called again"
            stack.pop()
        then: "it should throw an EmptyStackException"
            thrown EmptyStackException

    }

}
public class Stack {
    private Integer value = null;

    public void push(int value) {
        this.value = value;
    }


    public int pop() {
        return value;
    }

}
```

```
class StackSpec extends Specification {
    def "Calling push(5), pop() should return 5. Calling pop() again
         should throw an EmptyStackException"() {
        given: "we call push(5) on a new instance"
            def stack = new Stack()
            stack.push(5)
        expect: "when pop() is called, it should return 5"
            stack.pop() == 5
        when: "pop() is called again"
            stack.pop()
        then: "it should throw an EmptyStackException"
            thrown EmptyStackException

    }
```

Run: StackSpecification

Tests failed: 2 of 2 tests – 48 ms

Test Results	48 ms
StackSpecification	48 ms
Calling pop() on a new Stack	33 ms
Calling push(5), pop() should	15 ms

```
Expected exception of type 'java.util.EmptyStackException', but no exception was thrown

    at org.spockframework.lang.SpecInternals.checkExceptionThrown(SpecInternals.java:81)
    at org.spockframework.lang.SpecInternals.thrownImpl(SpecInternals.java:68)
    at com.grokspock.devnexus2021.StackSpecification.Calling pop() on a new Stack should throw an EmptyStackException(StackSpecifica
```

```
    public void push(int value) {
        this.value = value;
    }


    public int pop() {
        return value;
    }

}
```

```
class StackSpec extends Specification {
    def "Calling push(5), pop() should return 5. Calling pop() again
         should throw an EmptyStackException"() {
        given: "we call push(5) on a new instance"
            def stack = new Stack()
            stack.push(5)
        expect: "when pop() is called, it should return 5"
            stack.pop() == 5
        when: "pop() is called again"
            stack.pop()
        then: "it should throw an EmptyStackException"
            thrown EmptyStackException

    }

}
public class Stack {
    private Integer value = null;

    public void push(int value) {
        this.value = value;
    }


    public int pop() {
        return value;
    }

}
```

```
class StackSpec extends Specification {
    def "Calling push(5), pop() should return 5. Calling pop() again
         should throw an EmptyStackException"() {
        given: "we call push(5) on a new instance"
            def stack = new Stack()
            stack.push(5)
        expect: "when pop() is called, it should return 5"
            stack.pop() == 5
        when: "pop() is called again"
            stack.pop()
        then: "it should throw an EmptyStackException"
            thrown EmptyStackException

    }

}
public class Stack {
    private Integer value = null;

    public int pop() {
        return value;
    }

}
```

```
class StackSpec extends Specification {
    def "Calling push(5), pop() should return 5. Calling pop() again
         should throw an EmptyStackException"() {
        given: "we call push(5) on a new instance""
            def stack = new Stack()
            stack.push(5)
        expect: "when pop() is called, it should return 5"
            stack.pop() == 5
        when: "pop() is called again"
            stack.pop()
        then: "it should throw an EmptyStackException"
            thrown EmptyStackException

    }

}
public class Stack {
    private Integer value = null;

    public int pop() {
        if (value == null) {
            throw new EmptyStackException();
        }

        return value;
    }

}
```

```
class StackSpec extends Specification {
    def "Calling push(5), pop() should return 5. Calling pop() again
         should throw an EmptyStackException"() {
        given: "we call push(5) on a new instance""
            def stack = new Stack()
            stack.push(5)
        expect: "when pop() is called, it should return 5"
            stack.pop() == 5
        when: "pop() is called again"
            stack.pop()
        then: "it should throw an EmptyStackException"
            thrown EmptyStackException

    }

}
public class Stack {
    private Integer value = null;

    public int pop() {
        if (value == null) {
            throw new EmptyStackException();
        }

        int topValue = value;
        value = null;
        return topValue;
    }
```

```
class StackSpec extends Specification {
    def "Calling push(5), pop() should return 5. Calling pop() again
         should throw an EmptyStackException"() {
        given: "we call push(5) on a new instance"
            def stack = new Stack()
            stack.push(5)
        expect: "when pop() is called, it should return 5"
            stack.pop() == 5
        when: "pop() is called again"
            stack.pop()
        then: "it should throw an EmptyStackException"
            thrown EmptyStackException

    }
```

Run: StackSpecification.Calling pop() after push(5) should return 5...

Tests passed: 1 of 1 test – 29 ms

Test Results	29 ms
com.grokspock.devnexus2021.	29 ms
Calling pop() after push(5) s	29 ms

```
BUILD SUCCESSFUL in 2s
5 actionable tasks: 1 executed, 4 up-to-date
3:55:49 AM: Task execution finished ':test --tests "com.grokspock.devnexus2021.StackSpecification.Calling pop() after push(5) should return 5* Calling pop() ag
```

```
    private Integer value = null;

    public int pop() {
        if (value == null) {
            throw new EmptyStackException();
        }

        int topValue = value;
        value = null;
        return topValue;
    }
```

```
class StackSpec extends Specification {
    def "Calling push(5), pop() should return 5. Calling pop() again
         should throw an EmptyStackException"() {
        given: "we call push(5) on a new instance"
            def stack = new Stack()
            stack.push(5)
        expect: "when pop() is called, it should return 5"
            stack.pop() == 5
        when: "pop() is called again"
            stack.pop()
        then: "it should throw an EmptyStackException"
            thrown EmptyStackException

    }
}
```

Refactor!

```
public class Stack {
    private Integer value = null;

    public int pop() {
        if (value == null) {
            throw new EmptyStackException();
        }

        int topValue = value;
        value = null;
        return topValue;
    }
```

File | /git/tdd-a-stack/build/spock-reports/com.grokspock.devnexus2021.StackSpecification.html

Report for com.grokspock.devnexus2021.StackSpecification

<< Back

Summary:

Created on Sun Jan 24 13:37:17 EST 2021 by bth0624

Executed features	Passed	Failures	Errors	Skipped	Success rate	Time
2	2	0	0	0	100.0%	0.025 seconds

Features:

- Calling pop() on a new Stack should throw an EmptyStackException
- Calling pop() after push(5) should return 5. Calling pop() again should throw an EmptyStackException

Calling pop() on a new Stack should throw an EmptyStackException

Return

Given: a new instance

When: pop() is called

Then: it should throw an EmptyStackException

Calling pop() after push(5) should return 5. Calling pop() again should throw an EmptyStackException

Return

Given: a new Stack instance

Expect: calling pop should return 5

When: pop() is called

Then: ----

Generated by Athaydes Spock Reports

Stack Scenarios

Given a new Stack:

* Calling pop should throw an EmptyStackException.
* Calling pop after pushing 5, should return 5. Calling pop again should throw an EmptyStackException.
* Pushing 2 and 3, then calling pop() twice should return 3, 2.

```
class StackSpec extends Specification {
    def "Pushing 2 and 3, then calling pop() twice should return 3, 2."() {
        given: "a new instance"

        when: "2 and 3 are pushed"


        and: "pop() is called twice"


        then: "it should return 3, 2"



    }
}
```

```
public class Stack {
    private Integer value = null;

    public void push(int value) {
        this.value = value;
    }

    public int pop() {
        if (value == null) {
            throw new EmptyStackException();
        }

        int topValue = value;
```

```
class StackSpec extends Specification {
    def "Pushing 2 and 3, then calling pop() twice should return 3, 2."() {
        given: "a new instance"
            def stack = new Stack()
        when: "2 and 3 are pushed"
            stack.push(2)
            stack.push(3)
        and: "pop() is called twice"
            def firstValue = stack.pop()
            def secondValue = stack.pop()
        then: "it should return 3, 2"
            firstValue == 3
            secondValue == 2

    }
}
```

```
public class Stack {
    private Integer value = null;

    public void push(int value) {
        this.value = value;
    }

    public int pop() {
        if (value == null) {
            throw new EmptyStackException();
        }

        int topValue = value;
```

```groovy
class StackSpec extends Specification {
    def "Pushing 2 and 3, then calling pop() twice should return 3, 2."() {
        given: "a new instance"
            def stack = new Stack()
        when: "2 and 3 are pushed"
            stack.push(2)
            stack.push(3)
        and: "pop() is called twice"
            def firstValue = stack.pop()
            def secondValue = stack.pop()
        then: "it should return 3, 2"
            firstValue == 3
            secondValue == 2
```

Run: StackSpecification.Pushing 2 and 3, then calling pop (1)

Tests failed: 1 of 1 test – 48 ms

Test Results	48 ms
StackSpecification	48 ms
Pushing 2 and 3, then calling	48 ms

```
java.util.EmptyStackException Create breakpoint
    at com.grokspock.devnexus2021.Stack.pop(Stack.java:14)
    at com.grokspock.devnexus2021.StackSpecification.Pushing 2 and 3, then calling pop() twice should return 3, 2.(StackSpecification.groovy:45)
```

```java
    private Integer value = null;

    public void push(int value) {
        this.value = value;
    }

    public int pop() {
        if (value == null) {
            throw new EmptyStackException();
        }

        int topValue = value;
```

```
class StackSpec extends Specification {
    def "Pushing 2 and 3, then calling pop() twice should return 3, 2."() {
        given: "a new instance"
            def stack = new Stack()
        when: "2 and 3 are pushed"
            stack.push(2)
            stack.push(3)
        and: "pop() is called twice"
            def firstValue = stack.pop()
            def secondValue = stack.pop()
        then: "it should return 3, 2"
            firstValue == 3
            secondValue == 2

    }
}
```

```
public class Stack {
    private Integer value = null;

    public void push(int value) {
        this.value = value;
    }

    public int pop() {
        if (value == null) {
            throw new EmptyStackException();
        }

        int topValue = value;
```

```
class StackSpec extends Specification {
    def "Pushing 2 and 3, then calling pop() twice should return 3, 2."() {
        given: "a new instance"
            def stack = new Stack()
        when: "2 and 3 are pushed"
            stack.push(2)
            stack.push(3)
        and: "pop() is called twice"
            def firstValue = stack.pop()
            def secondValue = stack.pop()
        then: "it should return 3, 2"
            firstValue == 3
            secondValue == 2

    }
}
```

```
public class Stack {
    private List<Integer> value = new ArrayList<>();

    public void push(int value) {
        this.value = value;
    }

    public int pop() {
        if (value == null) {
            throw new EmptyStackException();
        }

        int topValue = value;
```

```
class StackSpec extends Specification {
    def "Pushing 2 and 3, then calling pop() twice should return 3, 2."() {
        given: "a new instance"
            def stack = new Stack()
        when: "2 and 3 are pushed"
            stack.push(2)
            stack.push(3)
        and: "pop() is called twice"
            def firstValue = stack.pop()
            def secondValue = stack.pop()
        then: "it should return 3, 2"
            firstValue == 3
            secondValue == 2

    }
}
```

```
public class Stack {
    private List<Integer> value = new ArrayList<>();

    public void push(int value) {
        this.value.add(value);
    }

    public int pop() {
        if (value == null) {
            throw new EmptyStackException();
        }

        int topValue = value;
```

```
class StackSpec extends Specification {
    def "Pushing 2 and 3, then calling pop() twice should return 3, 2."() {
        given: "a new instance"
            def stack = new Stack()
        when: "2 and 3 are pushed"
            stack.push(2)
            stack.push(3)
        and: "pop() is called twice"
            def firstValue = stack.pop()
            def secondValue = stack.pop()
        then: "it should return 3, 2"
            firstValue == 3
            secondValue == 2

    }
}
```

```
public class Stack {
    private List<Integer> value = new ArrayList<>();

    public void push(int value) { this.value.add(value); }

    public int pop() {
        if (value == null) {
            throw new EmptyStackException();
        }

        int topValue = value;
        value = null;
        return topValue;
```

```
class StackSpec extends Specification {
    def "Pushing 2 and 3, then calling pop() twice should return 3, 2."() {
        given: "a new instance"
            def stack = new Stack()
        when: "2 and 3 are pushed"
            stack.push(2)
            stack.push(3)
        and: "pop() is called twice"
            def firstValue = stack.pop()
            def secondValue = stack.pop()
        then: "it should return 3, 2"
            firstValue == 3
            secondValue == 2

    }
}
```

```
public class Stack {
    private List<Integer> value = new ArrayList<>();

    public void push(int value) { this.value.add(value); }

    public int pop() {
        if (value.isEmpty()) {
            throw new EmptyStackException();
        }

        return value.remove(value.size()-1);
    }
```

```
class StackSpec extends Specification {
    def "Pushing 2 and 3, then calling pop() twice should return 3, 2."() {
        given: "a new instance"
            def stack = new Stack()
        when: "2 and 3 are pushed"
            stack.push(2)
            stack.push(3)
        and: "pop() is called twice"
            def firstValue = stack.pop()
            def secondValue = stack.pop()
        then: "it should return 3, 2"
            firstValue == 3
```

Run: StackSpecification

Tests passed: 3 of 3 tests – 30 ms

Test Results	30 ms
StackSpecification	30 ms
Calling pop() on a new Stack	23 ms
Calling push(5), pop() should	6 ms
Pushing 2 and 3, then calling p	1 ms

```
Process finished with exit code 0
```

```
public class Stack {
    private List<Integer> value = new ArrayList<>();

    public void push(int value) { this.value.add(value); }

    public int pop() {
        if (value.isEmpty()) {
            throw new EmptyStackException();
        }

        return value.remove(value.size()-1);
    }
}
```

```
class StackSpec extends Specification {
    def "Pushing 2 and 3, then calling pop() twice should return 3, 2."() {
        given: "a new instance"
            def stack = new Stack()
        when: "2 and 3 are pushed"
            stack.push(2)
            stack.push(3)
        and: "pop() is called twice"
            def firstValue = stack.pop()
            def secondValue = stack.pop()
        then: "it should return 3, 2"
            firstValue == 3
            secondValue == 2
    }
}
```

Refactor!

```
public class Stack {
    private List<Integer> value = new ArrayList<>();

    public void push(int value) { this.value.add(value); }

    public int pop() {
        if (value.isEmpty()) {
            throw new EmptyStackException();
        }

        return value.remove(value.size()-1);
    }
```

```
class StackSpec extends Specification {
    def "Pushing 2 and 3, then calling pop() twice should return 3, 2."() {
        given: "a new instance"
            def stack = new Stack()
        when: "2 and 3 are pushed"
            stack.push(2)
            stack.push(3)
        and: "pop() is called twice"
            def firstValue = stack.pop()
            def secondValue = stack.pop()
        then: "it should return 3, 2"
            firstValue == 3
            secondValue == 2
    }
}
```

Refactor!

```
public class Stack {
    private List<Integer> values = new ArrayList<>();

    public void push(int value) { this.values.add(value); }

    public int pop() {
        if (values.isEmpty()) {
            throw new EmptyStackException();
        }

        return values.remove(values.size()-1);
    }
```

```
class StackSpec extends Specification {
    def "Pushing 2 and 3, then calling pop() twice should return 3, 2."() {
        given: "a new instance"
            def stack = new Stack()
        when: "2 and 3 are pushed"
            stack.push(2)
            stack.push(3)
        and: "pop() is called twice"
            def firstValue = stack.pop()
            def secondValue = stack.pop()
        then: "it should return 3, 2"
            firstValue == 3
            secondValue == 2
    }
}
```

Refactor!

```
public class Stack {
    private List<Integer> values = new ArrayList<>();

    public void push(int value) {
        this.values.add(value);
    }

    public int pop() {
        if (values.isEmpty()) {
            throw new EmptyStackException();
        }

        return values.remove(values.size()-1);
```

```
class StackSpec extends Specification {
    def "Pushing 2 and 3, then calling pop() twice should return 3, 2."() {
        given: "a new instance"
            def stack = new Stack()
        when: "2 and 3 are pushed"
            stack.push(2)
            stack.push(3)
        and: "pop() is called twice"
            def firstValue = stack.pop()
            def secondValue = stack.pop()
        then: "it should return 3, 2"
            firstValue == 3
            secondValue == 2
    }
}
```

Refactor!

```
public class Stack {
    private List<Integer> values = new ArrayList<>();

    public void push(int value) {
        values.add(value);
    }

    public int pop() {
        if (values.isEmpty()) {
            throw new EmptyStackException();
        }

        return values.remove(values.size()-1);
```

```
class StackSpec extends Specification {
    def "Pushing 2 and 3, then calling pop() twice should return 3, 2."() {
        given: "a new instance"
            def stack = new Stack()
        when: "2 and 3 are pushed"
            stack.push(2)
            stack.push(3)
        and: "pop() is called twice"
            def firstValue = stack.pop()
            def secondValue = stack.pop()
        then: "it should return 3, 2"
            firstValue == 3
```

Run: StackSpecification

Tests passed: 3 of 3 tests – 33 ms

Test Results	33 ms
StackSpecification	33 ms
Calling pop() on a new Stack	27 ms
Calling push(5), pop() should	6 ms
Pushing 2 and 3, then calling pop()	

```
Process finished with exit code 0
```

```
public class Stack {
    private List<Integer> values = new ArrayList<>();

    public void push(int value) {
        values.add(value);
    }

    public int pop() {
        if (values.isEmpty()) {
            throw new EmptyStackException();
        }

        return values.remove(values.size()-1);
}
```

Stack Scenarios

Given a new Stack:

* Calling pop should throw an EmptyStackException.
* Calling pop after pushing 5, should return 5. Calling pop again should throw an EmptyStackException.
* Pushing 2 and 3, then calling pop() twice should return 3, 2.

```
@Narrative("""
Like a stack of books on a table, a Stack is a data structure
that stores and returns data in a Last-in-First-out manner.

Methods:
  push() — Adds an item to the top of the stack.
  pop()  — Removes & returns top item. If it's empty, throws EmptyStackException.
""")


public class Stack {
    private List<Integer> values = new ArrayList<>();


    public void push(int value) {
        values.add(value);
    }

    public int pop() {
        if (values.isEmpty()) {
            throw new EmptyStackException();
        }

        return values.remove(values.size()-1);
    }

}
```

Report for com.grokspock.devnexus2021.StackSpecification

<< Back

Summary:

Created on Wed Feb 03 22:00:11 EST 2021 by bth0624

Executed features	Passed	Failures	Errors	Skipped	Success rate	Time
3	3	0	0	0	100.0%	0.037 seconds

Like a stack of books on a table, a Stack is a data structure that stores and returns data in a Last-in-First-out manner.

Methods:
push() — Adds an item to the top of the stack.
pop() — Removes & returns top item. If it's empty, throws EmptyStackException.

Narrative Text

Features:

- Calling pop() on a new Stack should throw an EmptyStackException
- Calling push(5), pop() should return 5. Calling pop() again should throw an EmptyStackException
- Pushing 2 and 3, then calling pop() twice should return 3, 2.

Calling pop() on a new Stack should throw an EmptyStackException

Return

Given: a new instance
When: pop() is called
Then: it should throw an EmptyStackException

Calling push(5), pop() should return 5. Calling pop() again should throw an EmptyStackException

Return

Given: we call push(5) on a new instance
Expect: when pop() is called, it should return 5
When: when calling pop() again
Then: it should throw an EmptyStackException

Return

Common Concerns

* TANSTAAFL
* Writing tests first is illogical. How can you test code that doesn't exist?
* How do I know what tests to write
* How do I know when I'm done?
* Rewriting tests if we change the implementation

```java
public class Stack {
    private List<Integer> values = new ArrayList<>();


    public void push(int value) {
        values.add(value);
    }

    public int pop() {
        if (values.isEmpty()) {
            throw new EmptyStackException();
        }

        return values.remove(values.size()-1);
    }

}
```

```java
public class Stack {
    private int[] values = new int[100000]; // Not unlimited
    private int valuesCount = 0;

    public void push(int value) {
        values[valuesCount++] = value;    // was values.add(value);
    }

    public int pop() {
        if (valuesCount == 0) {           // was values.isEmpty()
            throw new EmptyStackException();
        }

        return values[-valuesCount];      // was values.remove(values.size()-1)
    }
}
```

Run: StackSpecification

Tests passed: 3 of 3 tests – 37 ms

Test Results	37 ms
StackSpecification	37 ms
Calling pop() on a new Stack	24 ms
Calling push(5), pop() shoulc	10 ms
Pushing 2 and 3, then calling p	3 ms

```
Process finished with exit code 0
```

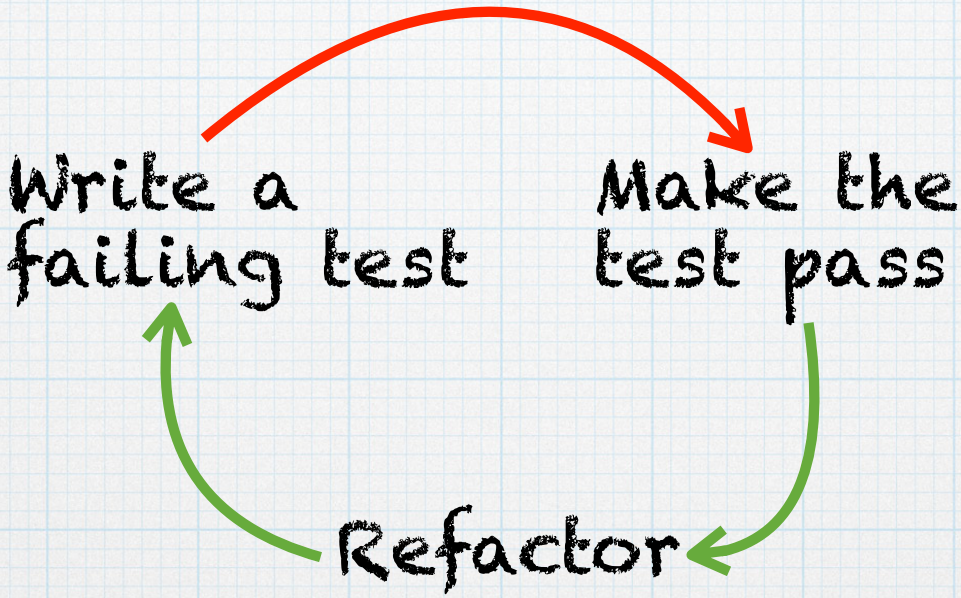
Goals

* Introducing TDD
* Succeeding with TDD
* Doing TDD in the Real World
* Summary

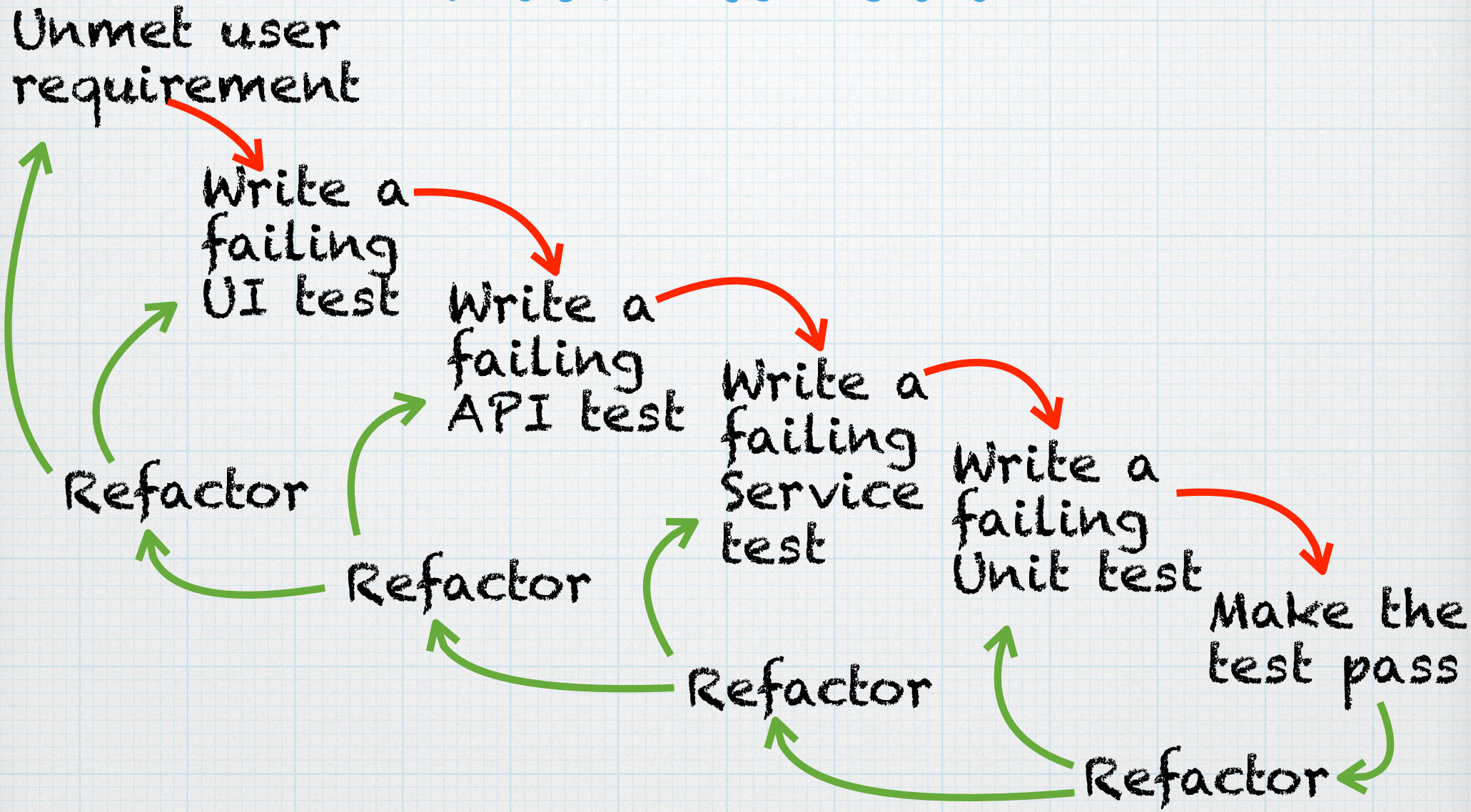
Doing TDD in the Real World

* Holistic TDD
* Practice writing unit specs
* Defect Driven Testing

Extreme TDD



Extreme TDD


Unmet user requirement
Write a failing UI test
Write a failing API test
Write a failing Service test
Write a failing Unit test
Make the test pass
Refactor
Refactor
Refactor
Refactor

Doing TDD in the Real World

* Holistic TDD
* Practice writing unit specs
* Defect Driven Testing

Practice writing unit specs

End-To-End Tests

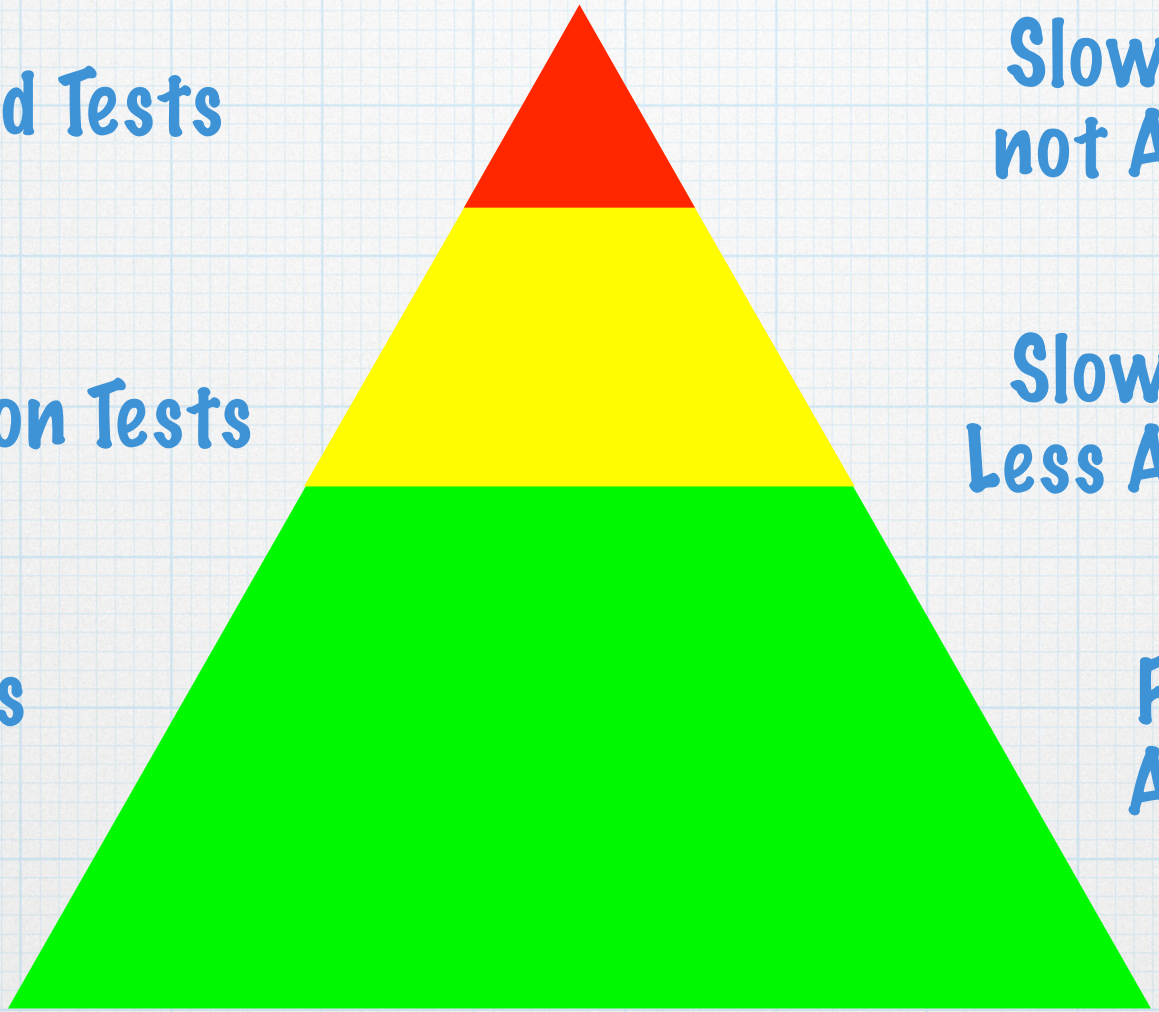
Slowest and not Accurate

Integration Tests

Slower and Less Accurate

Unit Tests

Fast and Accurate



Practice writing unit specs

* Search for "coding katas"
  * codewars.com or codingdojo.org
* Read the kata and look for requirements
  * Write scenarios for each requirement
  * Write executable specs to make all the scenarios pass
  * Remember to refactor!
* When you get comfortable with this, try it on something small at work.

Doing TDD in the Real World

* Holistic TDD
* Practice writing unit specs
* Defect Driven Testing

Defect-Driven Testing

* When you have a bug to fix, especially in legacy code, do the following:
  * Find the cause of the problem
  * Write a test that fails because of the bug
  * Fix the problem
  * Watch the test pass
  * Check both sets of code into your revision control system

Goals

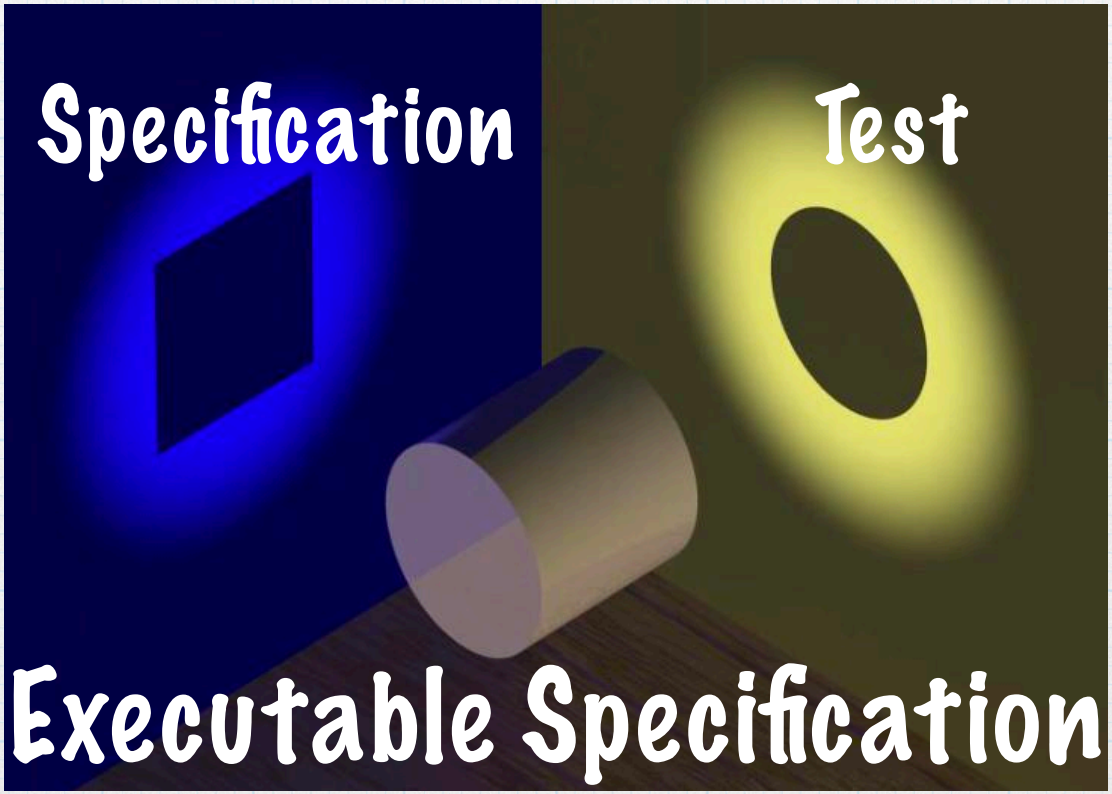
* Delivering Better Software, Faster with TDD
* Succeeding with TDD
* Doing TDD in the Real World
* Summary

TDD Benefits

* Better understanding of requirements
  * Incomplete or conflicting requirements are detected early
* Increased code quality: easily maintainable, highly cohesive, and loosely coupled
* Living documentation that is accurate and current; usable by business, other teams, and other devs on your team

Remember...

To succeed with TDD, write Executable Specifications and run them as Tests!



Thank you!

Live Long and Prosper