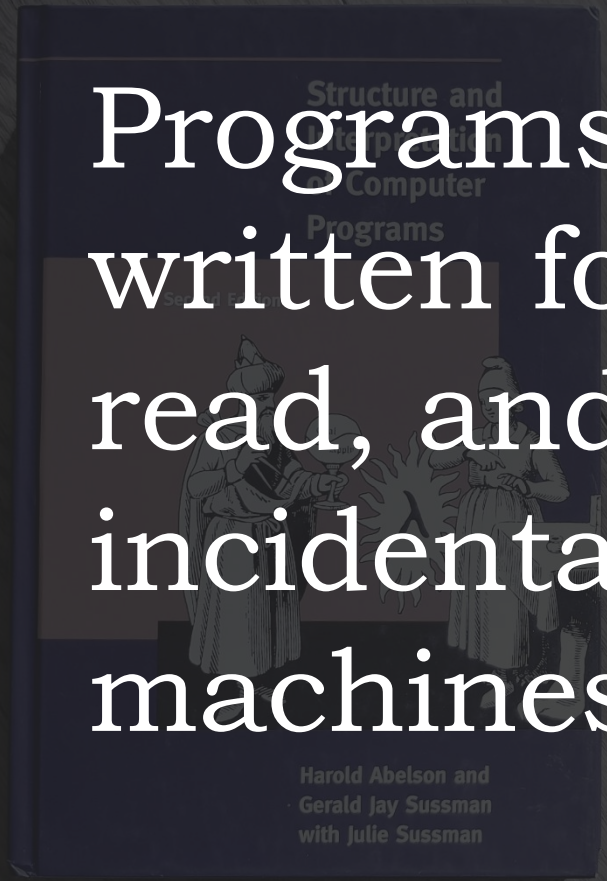
Structure and Interpretation of Test Cases

@KevlinHenney

Structure and Interpretation of Computer Programs

Second Edition

Harold Abelson and
Gerald Jay Sussman
with Julie Sussman



Programs must be written for people to read, and only incidentally for machines to execute.

The programmer in me made unit testing more about applying and exercising frameworks.

I had essentially reduced my concept of unit testing to the basic mechanics of exercising [unit testing frameworks] to verify the behavior of my classes.

My mindset had me thinking far too narrowly about what it meant to write good unit tests.

Tod Golding

Tapping into Testing Nirvana

good unit tests

GUTs

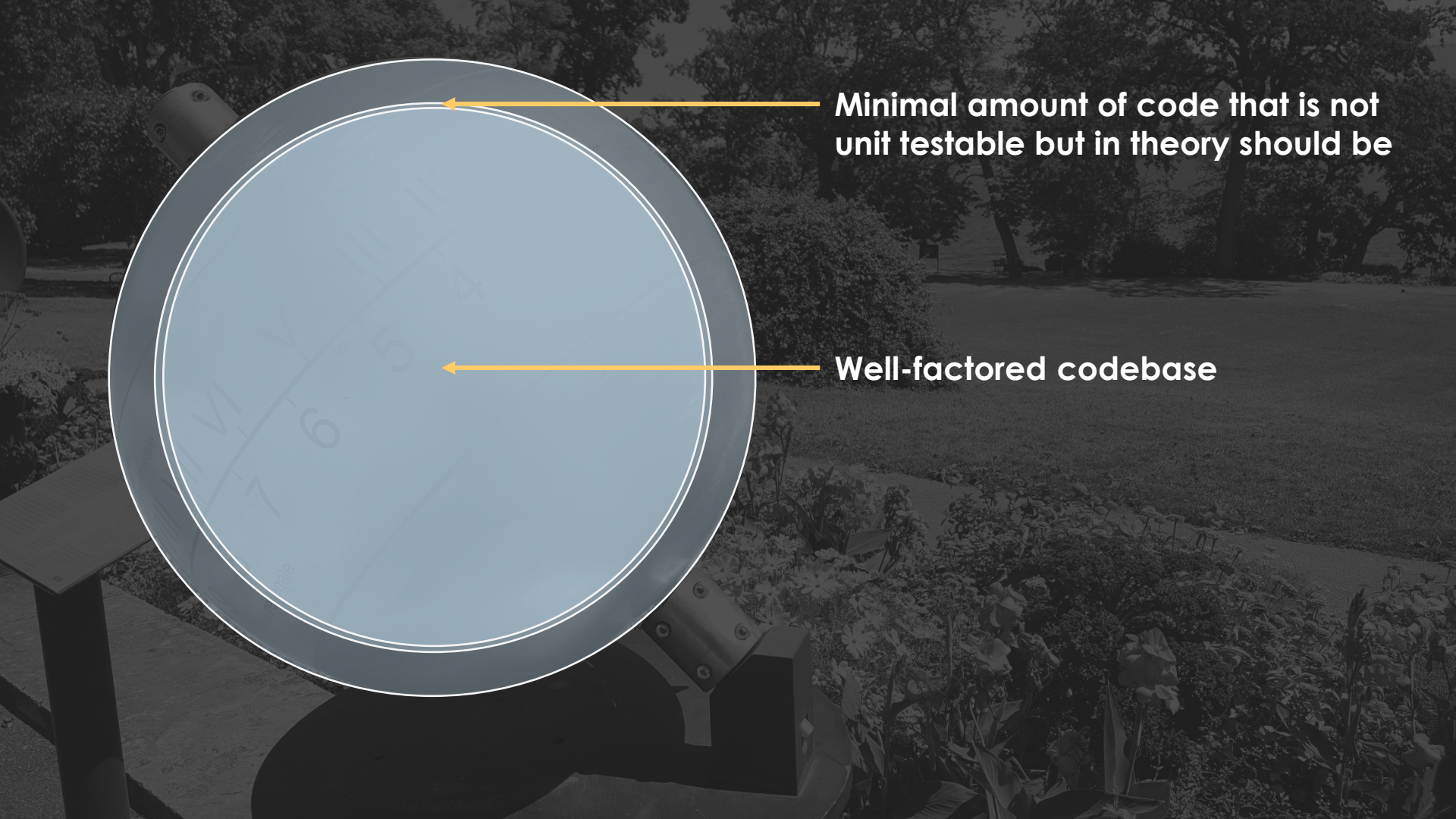
A unit test is a test of behaviour whose success or failure is wholly determined by the correctness of the test and the correctness of the unit under test.

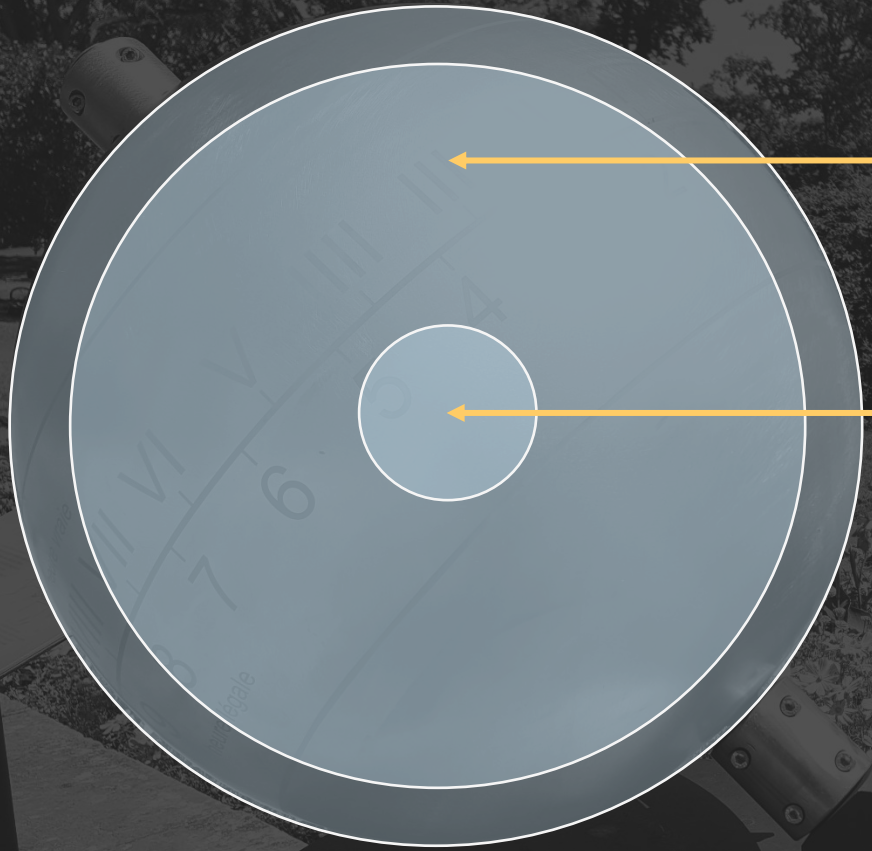
Kevlin Henney

theregister.co.uk/2007/07/28/what_are_your_units/


II III IIII V VI VII VIII
2 4 5 6 7 8
heure légale

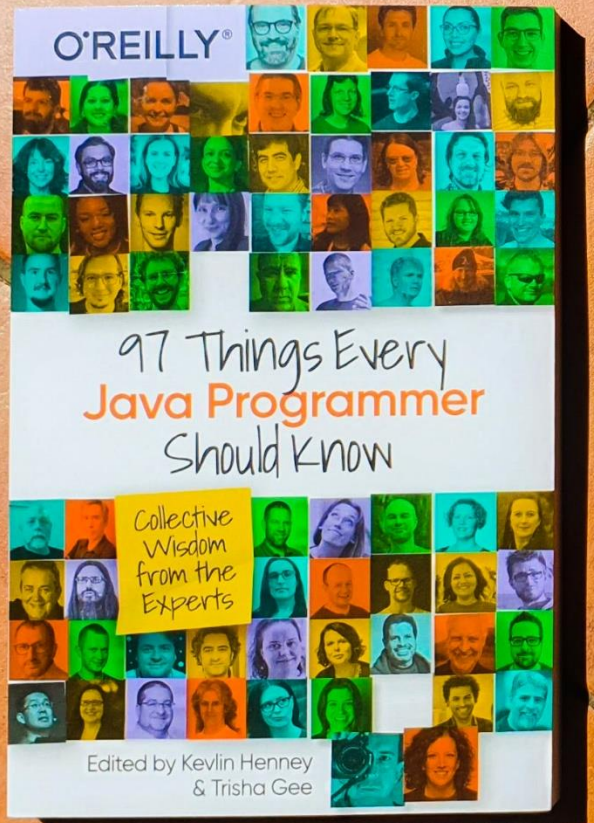

Necessarily not unit testable
Should be unit testable... but isn't
Unit testable

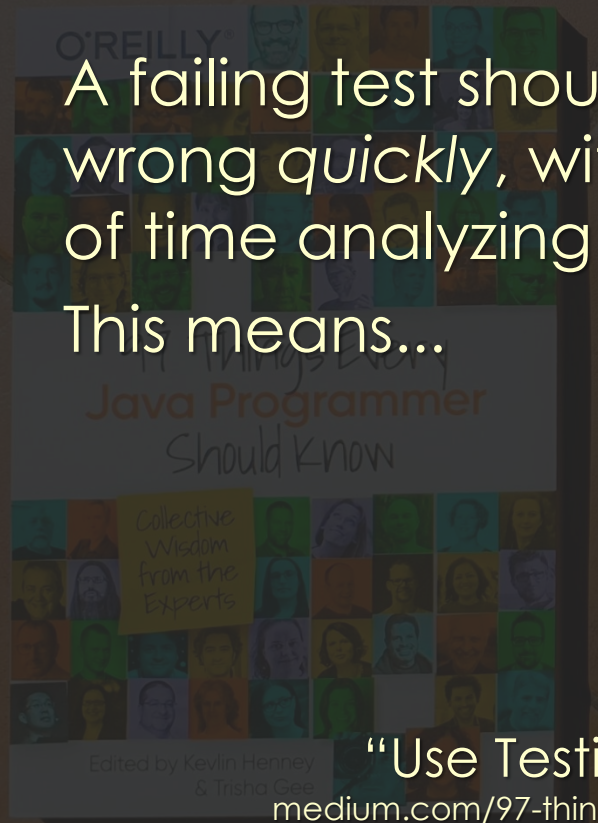

Minimal amount of code that is not unit testable but in theory should be
Well-factored codebase



Codebase with high coupling to code with dependencies across boundaries of trust and control

Minimal amount of code that is unit testable


O'REILLY®
97 Things Every
Java Programmer
Should Know
Collective Wisdom from the Experts
Edited by Kevlin Henney
& Trisha Gee



A failing test should tell you *exactly* what is wrong *quickly*, without you having to spend a lot of time analyzing the failure.

This means...

Marit van Dijk

"Use Testing to Develop Better Software Faster"

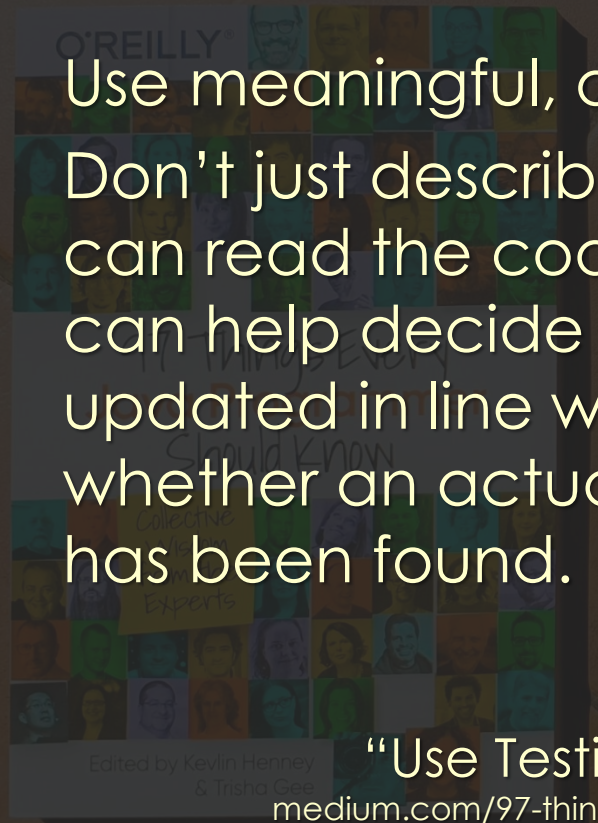
medium.com/97-things/use-testing-to-develop-better-software-faster-9dd2616543d3

Each test should test one thing.

Marit van Dijk

"Use Testing to Develop Better Software Faster"

medium.com/97-things/use-testing-to-develop-better-software-faster-9dd2616543d3



Use meaningful, descriptive names.

Don't just describe what the test does either (we can read the code), tell us why it does this. This can help decide whether a test should be updated in line with changed functionality or whether an actual failure that should be fixed has been found.

Marit van Dijk

"Use Testing to Develop Better Software Faster"

medium.com/97-things/use-testing-to-develop-better-software-faster-9dd2616543d3

Never trust a test you haven't seen fail.

Marit van Dijk

"Use Testing to Develop Better Software Faster"

medium.com/97-things/use-testing-to-develop-better-software-faster-9dd2616543d3

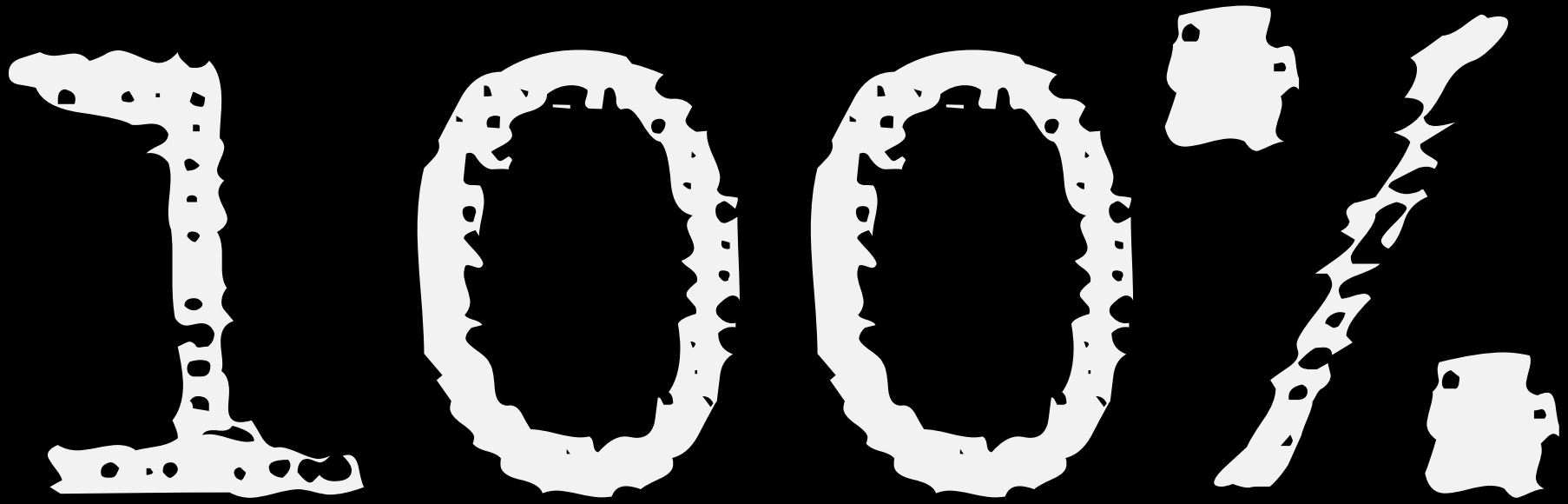
I expect a high level of coverage.
Sometimes managers require one.
There's a subtle difference.

Brian Marick

martinfowler.com/bliki/TestCoverage.html

coverage

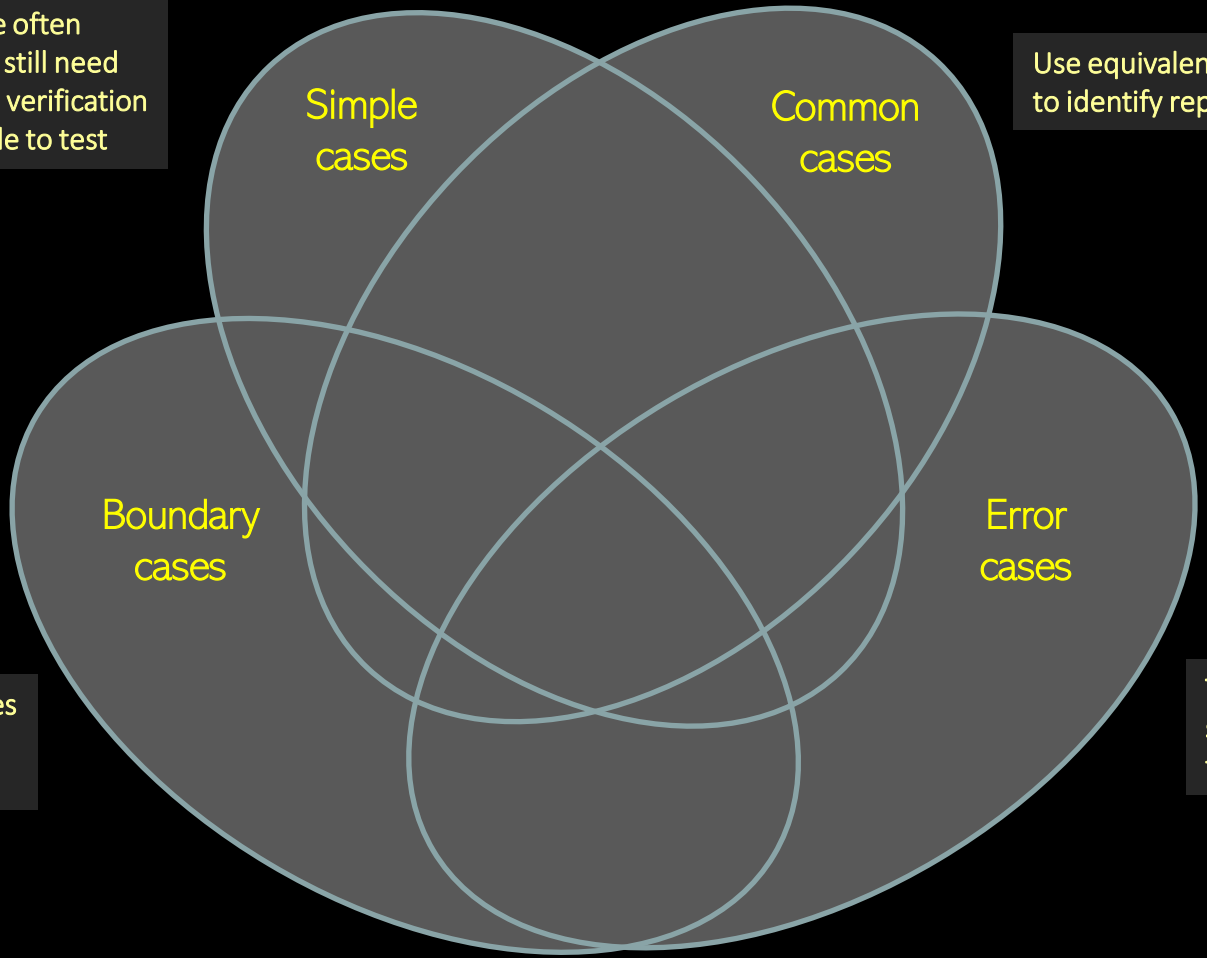
statement coverage


100%


DOJO

function coverage
statement coverage
branch coverage
loop coverage
condition coverage
multiple condition cover
path coverage
parameter value covera

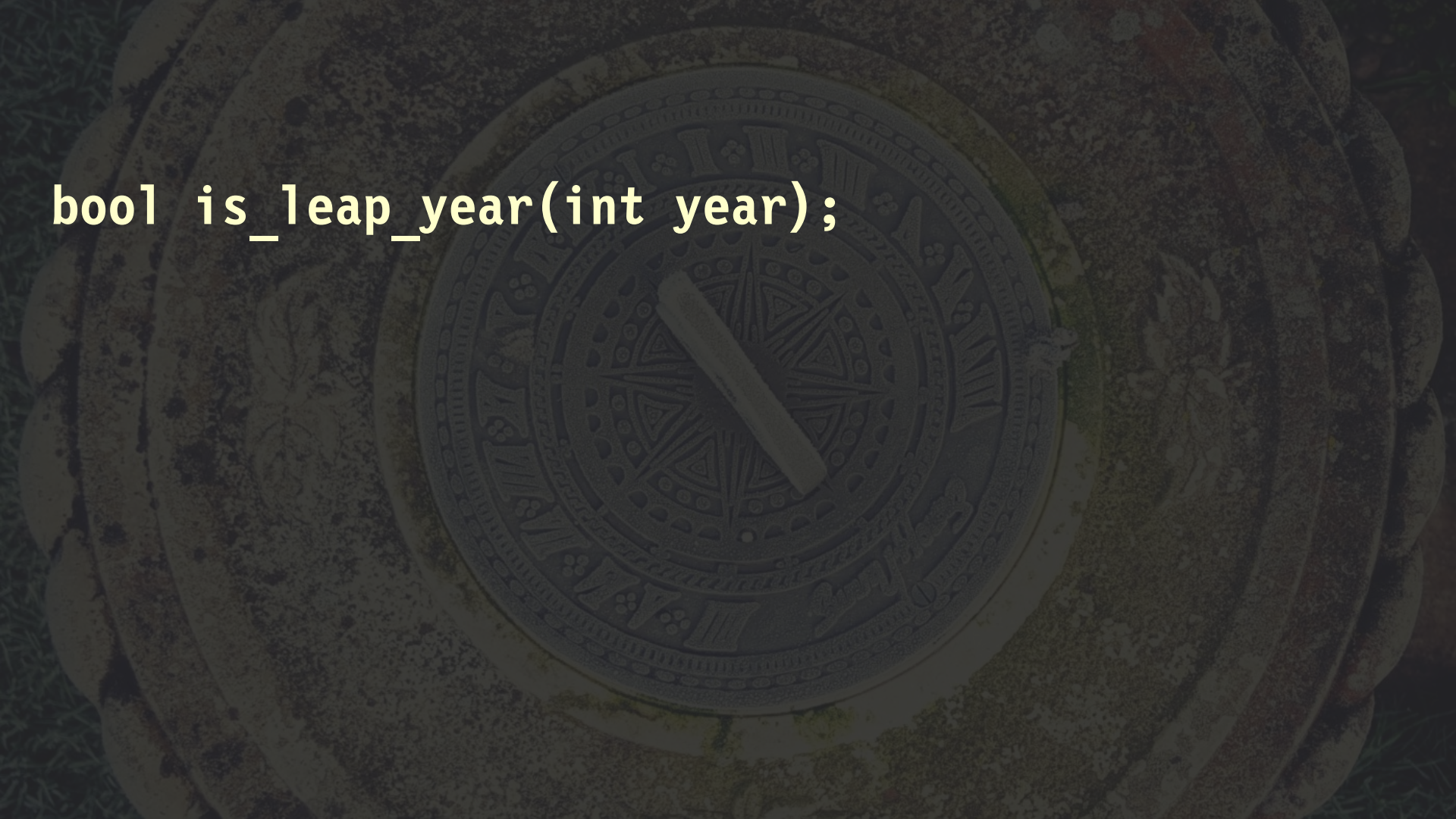
Simple cases are often overlooked, but still need clarification and verification — and are simple to test



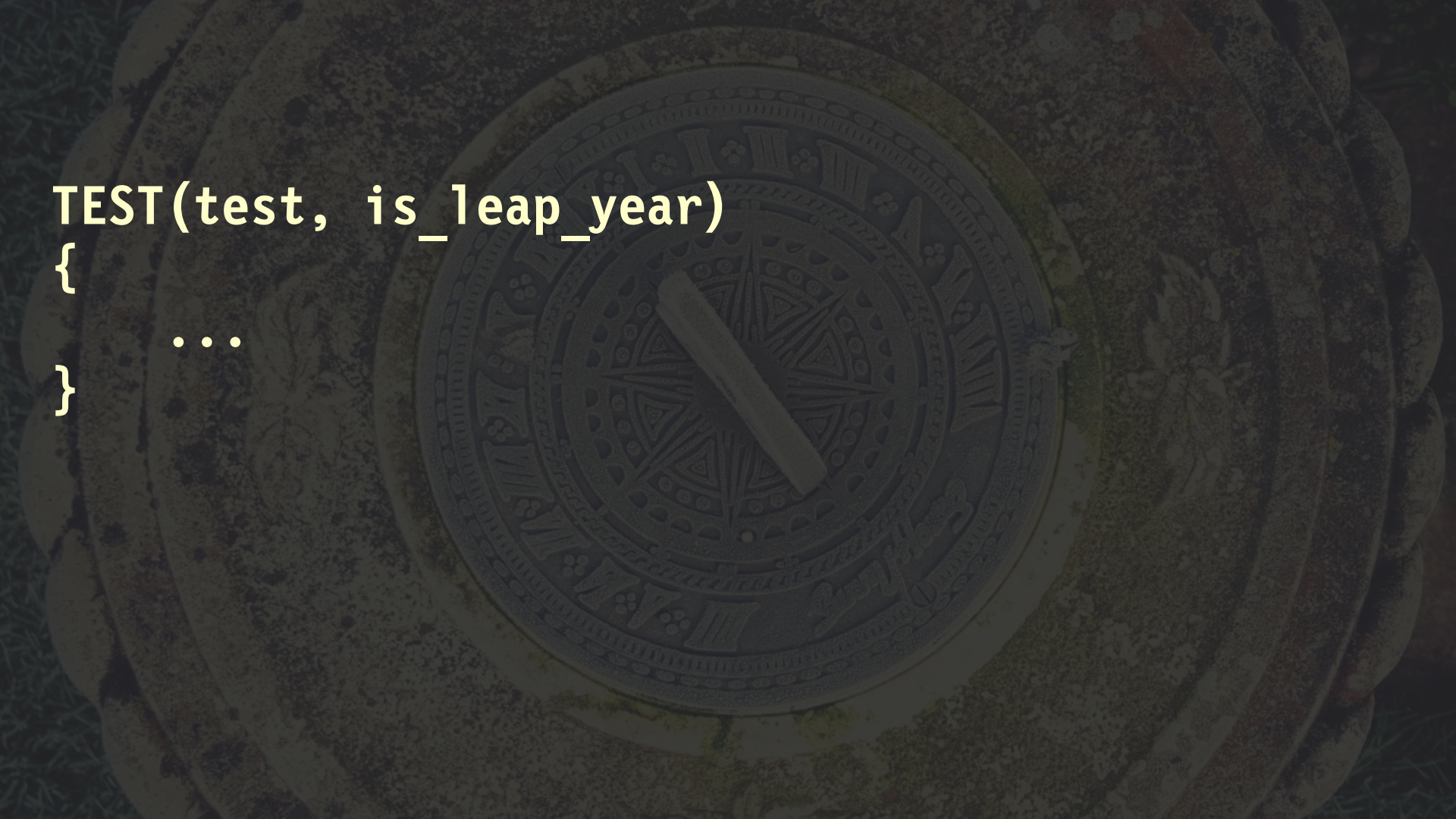
Use equivalence partitioning to identify representative data

Test around the edges — thresholds, nulls, empty cases, etc.

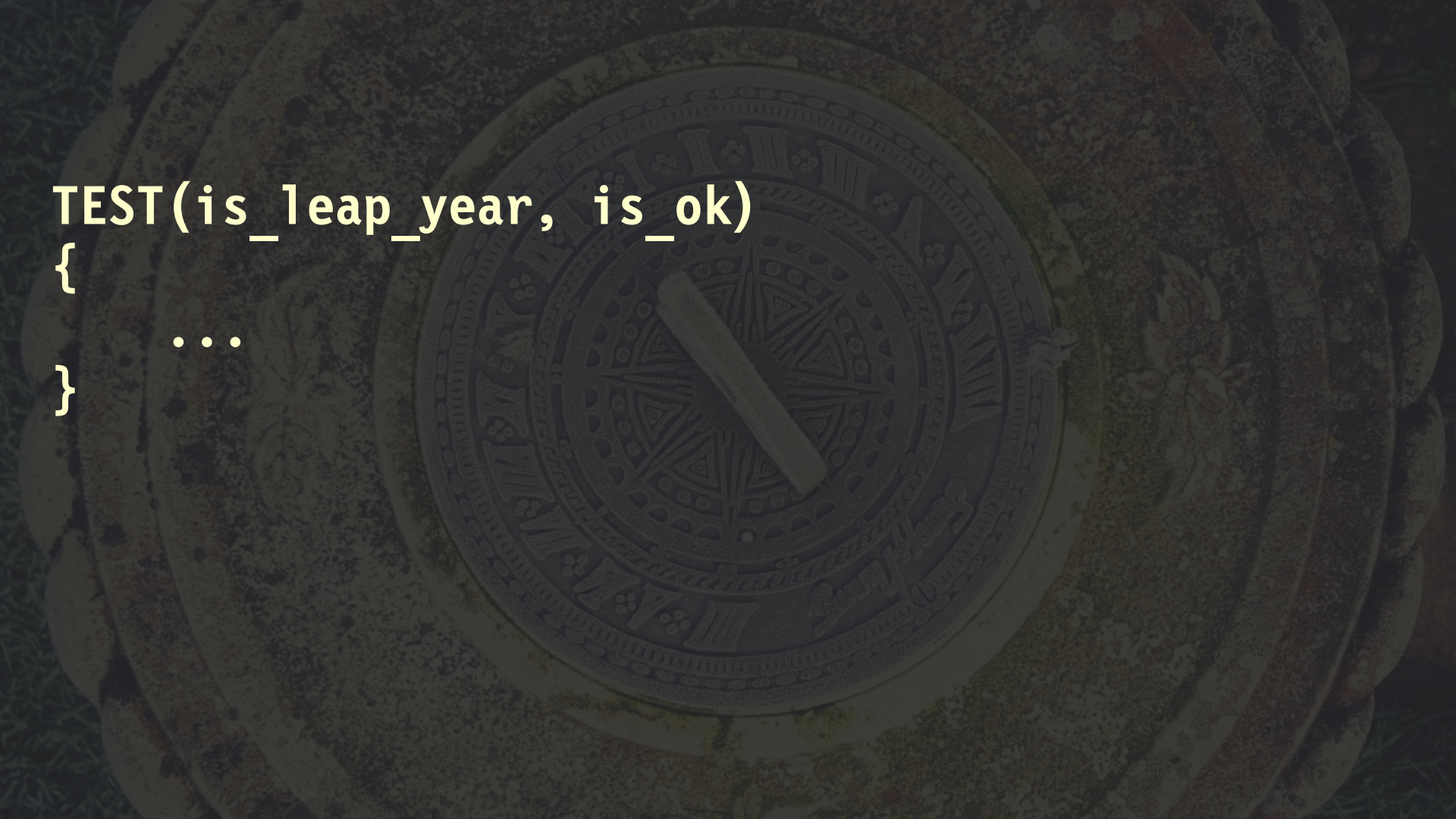
Test the rainy-day scenarios as well as the happy paths



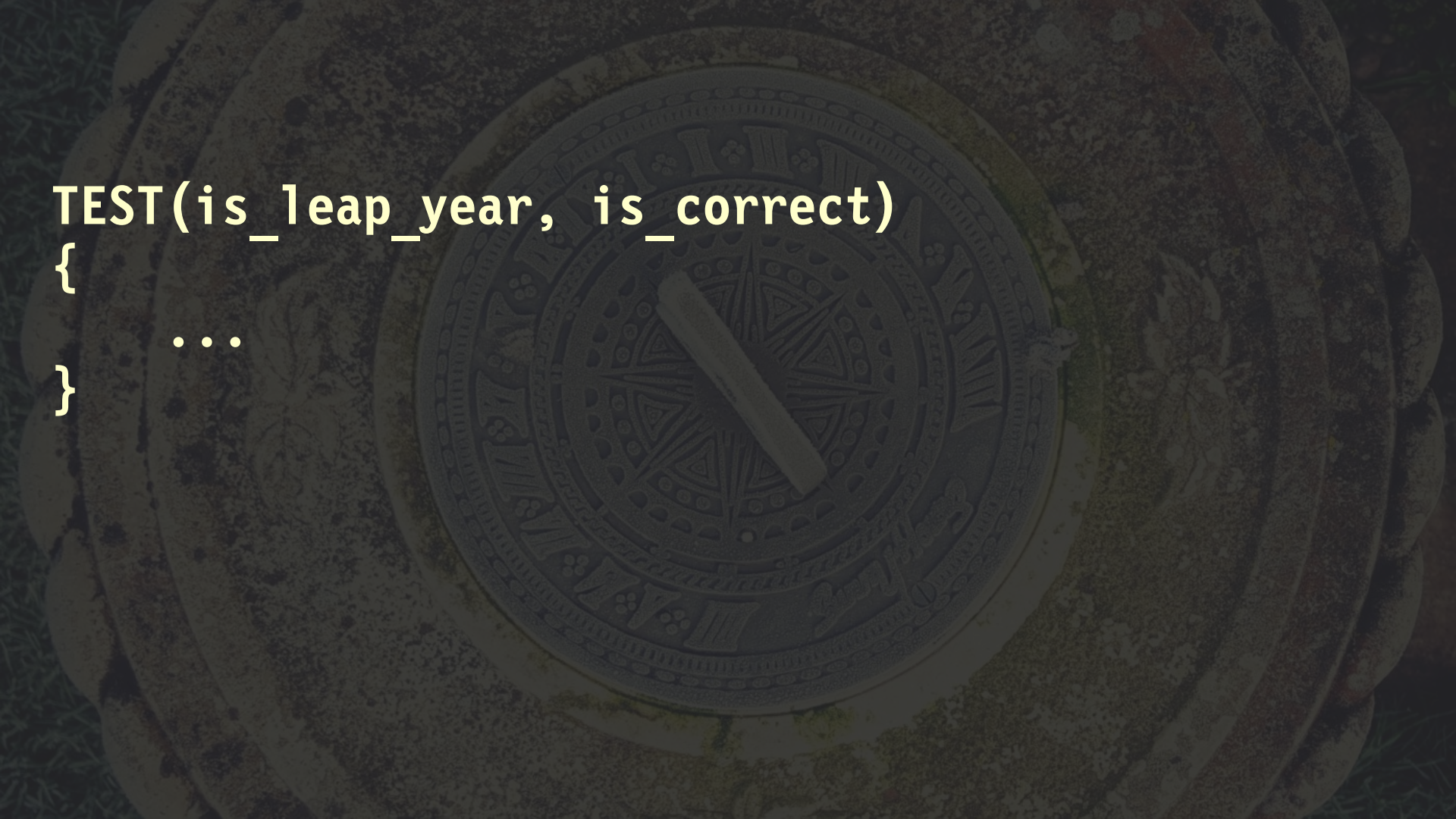
```
bool is_leap_year(int year);
```



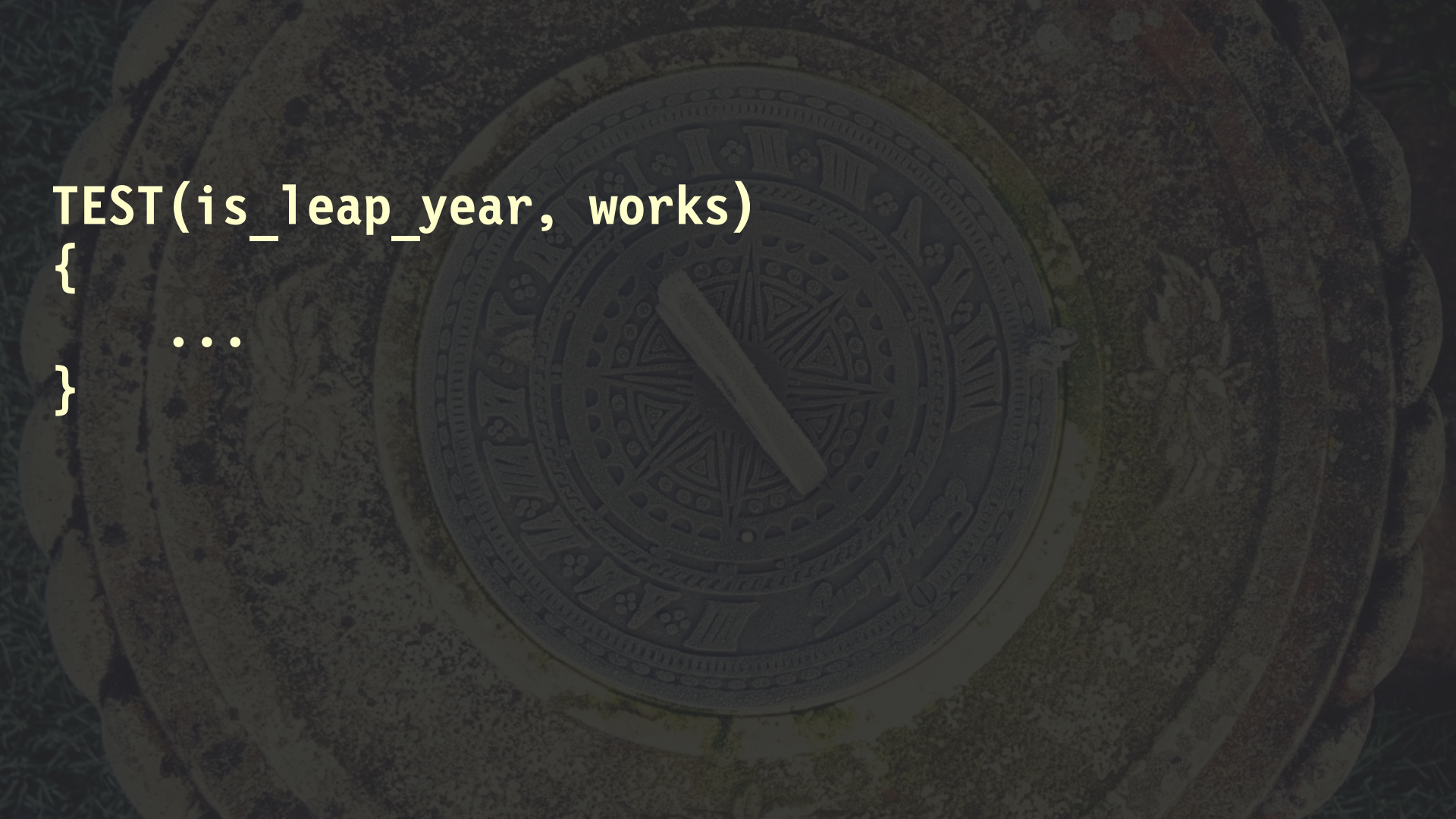
```
TEST(test, is_leap_year)
{
    ...
}
```



```
TEST(is_leap_year, is_ok)
{
    ...
}
```



```
TEST(is_leap_year, is_correct)
{
    ...
}
```



```
TEST(is_leap_year, works)
{
    ...
}
```

```
TEST(is_leap_year, works_as_expected)
{
    ...
}
```

```
TEST(is_leap_year, 1)
{
    ...
}
TEST(is_leap_year, 2)
{
    ...
}
```

```
TEST(is_leap_year, leap_years)
{
    ...
}
TEST(is_leap_year, non_leap_years)
{
    ...
}
```

```
TEST(is_leap_year, leap_years)
{
    ASSERT_TRUE(is_leap_year(2020));
    ASSERT_TRUE(is_leap_year(2000));
}
TEST(is_leap_year, non_leap_years)
{
    ...
}
```

```
TEST(is_leap_year, 2020) ...
TEST(is_leap_year, 2000) ...
TEST(is_leap_year, 2021) ...
TEST(is_leap_year, 1900) ...
```

```
TEST(is_leap_year, 2020_is_a_leap_year) ...
TEST(is_leap_year, 2000_is_a_leap_year) ...
TEST(is_leap_year, 2021_is_not_a_leap_year) ...
TEST(is_leap_year, 1900_is_not_a_leap_year) ...
```

```
TEST(is_leap_year, 2016_is_a_leap_year) ...
TEST(is_leap_year, 2400_is_a_leap_year) ...
TEST(is_leap_year, 2019_is_not_a_leap_year) ...
TEST(is_leap_year, 2100_is_not_a_leap_year) ...
```

```
TEST(A_year_divisible_by_4, is_a_leap_year) ...
TEST(A_year_divisible_by_400, is_a_leap_year) ...
TEST(A_year_not_divisible_by_4, is_not_a_leap_year
TEST(A_year_divisible_by_100, is_not_a_leap_year)
```

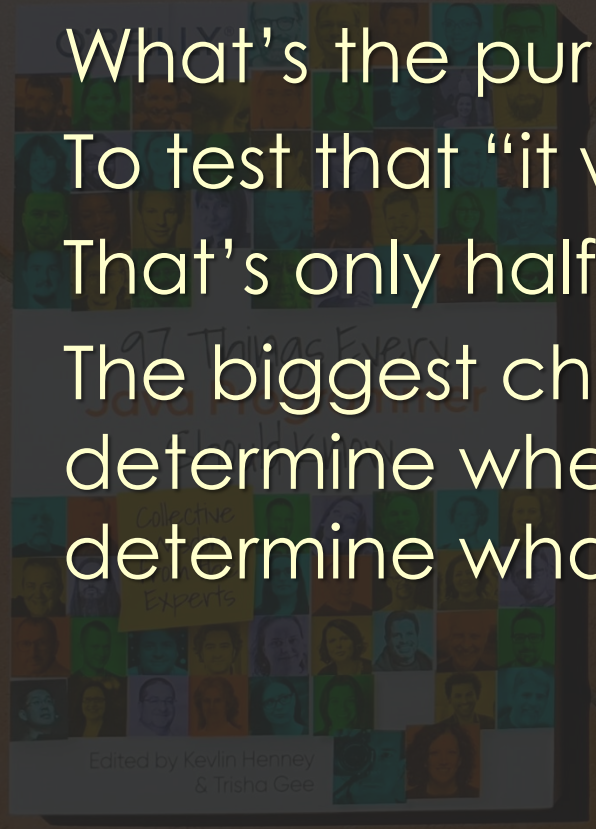
```
TEST(
    A_year_divisible_by_4,
    is_a_leap_year) ...
TEST(
    A_year_divisible_by_400,
    is_a_leap_year) ...
TEST(
    A_year_not_divisible_by_4,
    is_not_a_leap_year) ...
TEST(
    A_year_divisible_by_100,
    is_not_a_leap_year) ...
```

```
TEST(
    A_year_not_divisible_by_4,
    is_not_a_leap_year) ...
TEST(
    A_year_divisible_by_4,
    is_a_leap_year) ...
TEST(
    A_year_divisible_by_100,
    is_not_a_leap_year) ...
TEST(
    A_year_divisible_by_400,
    is_a_leap_year) ...
```

```
TEST(
    A_year_not_divisible_by_4,
    is_not_a_leap_year) ...
TEST(
    A_year_divisible_by_4_but_not_by_100,
    is_a_leap_year) ...
TEST(
    A_year_divisible_by_100_but_not_by_400,
    is_not_a_leap_year) ...
TEST(
    A_year_divisible_by_400,
    is_a_leap_year) ...
```

```
TEST(
    A_year_is_not_a_leap_year,
    if_it_is_not_divisible_by_4) ...
TEST(
    A_year_is_a_leap_year,
    if_it_is_divisible_4_but_not_by_100) ...
TEST(
    A_year_is_not_a_leap_year,
    if_it_is_divisible_by_100_but_not_by_400) ...
TEST(
    A_year_is_a_leap_year,
    if_it_is_divisible_by_400) ...
```

```
TEST(
    A year is not a leap year,
    if it is not divisible by 4) ...
TEST(
    A year is a leap year,
    if it is divisible 4 but not by 100) ...
TEST(
    A year is not a leap year,
    if it is divisible by 100 but not by 400) ...
TEST(
    A year is a leap year,
    if it is divisible by 400) ...
```



What's the purpose of your test?

To test that "it works"?

That's only half the story.

The biggest challenge in code is not to determine whether "it works", but to determine what "it works" means.

Kevlin Henney
"Program with GUTs"
medium.com/97-things/program-with-guts-828e69dd8e15

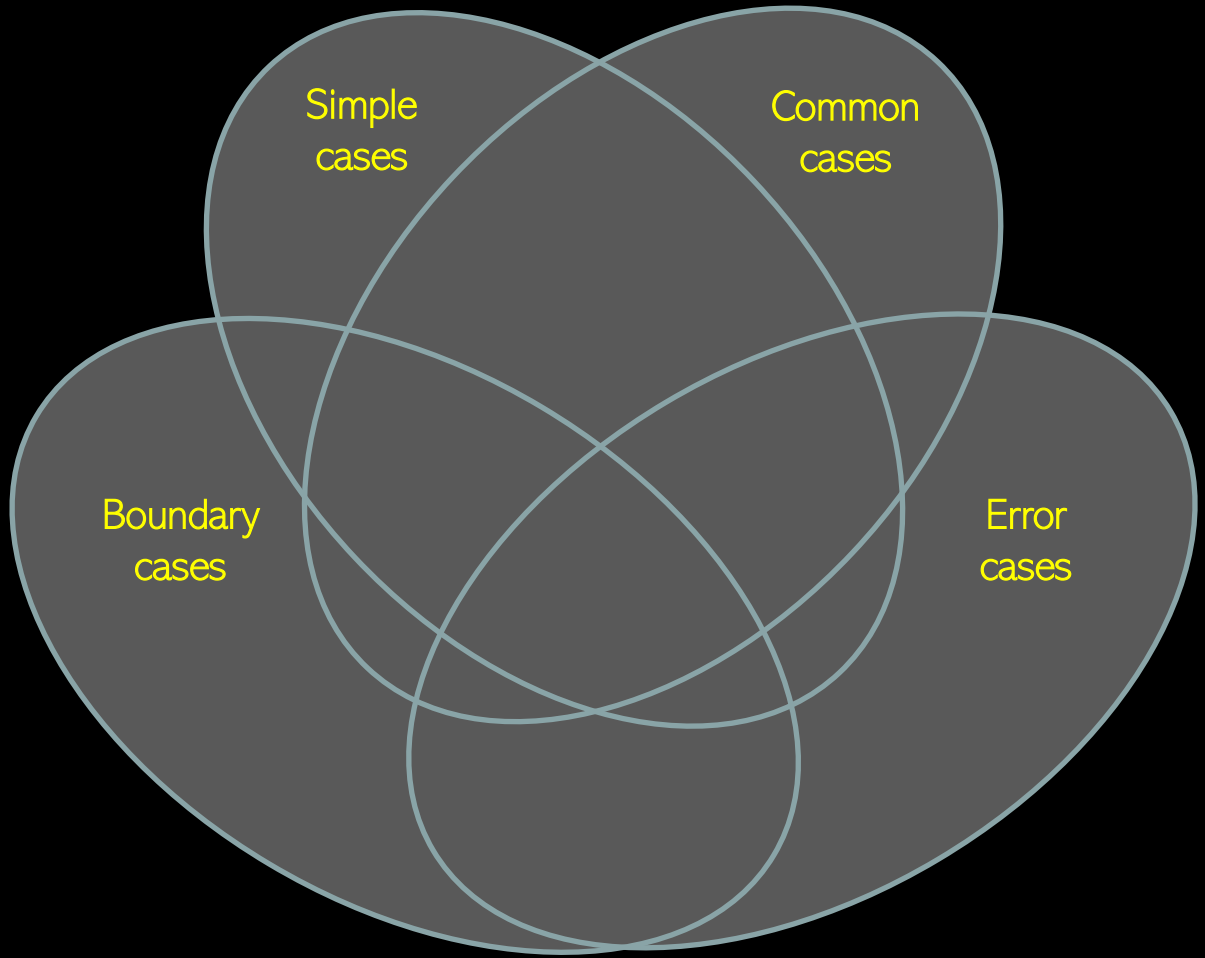
```
TEST(
    A year is not a leap year,
    if it is not divisible by 4) ...
TEST(
    A year is a leap year,
    if it is divisible 4 but not by 100) ...
TEST(
    A year is not a leap year,
    if it is divisible by 100 but not by 400) ...
TEST(
    A year is a leap year,
    if it is divisible by 400) ...
```

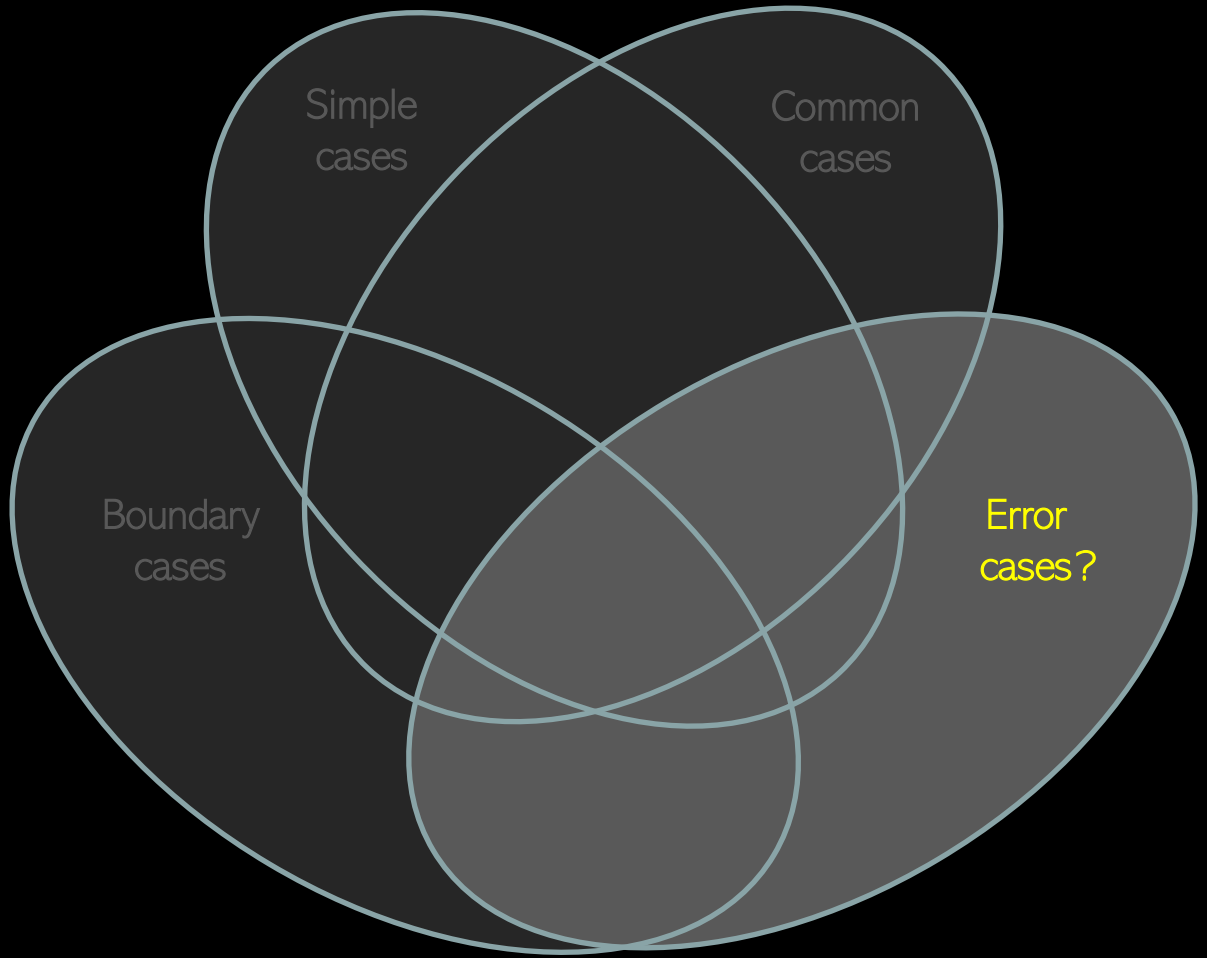
```
bool is_leap_year(int year)
{
    return
        year % 4 == 0 &&
        year % 100 != 0 ||
        year % 400 == 0;
}
```

```
TEST(
    A_year_is_not_a_leap_year,
    if_it_is_not_divisible_by_4) ...
TEST(
    A_year_is_a_leap_year,
    if_it_is_divisible_4_but_not_by_100) ...
TEST(
    A_year_is_not_a_leap_year,
    if_it_is_divisible_by_100_but_not_by_400) ...
TEST(
    A_year_is_a_leap_year,
    if_it_is_divisible_by_400) ...
```

```
bool is_leap_year(int year)
{
    return year % 4 == 0;
}
```

```
TEST(
    A_year_is_not_a_leap_year,
    if_it_is_not_divisible_by_4) ...
TEST(
    A_year_is_a_leap_year,
    if_it_is_divisible_4_but_not_by_100) ...
TEST(
    A_year_is_not_a_leap_year,
    if_it_is_divisible_by_100_but_not_by_400) ...
TEST(
    A_year_is_a_leap_year,
    if_it_is_divisible_by_400) ...
```

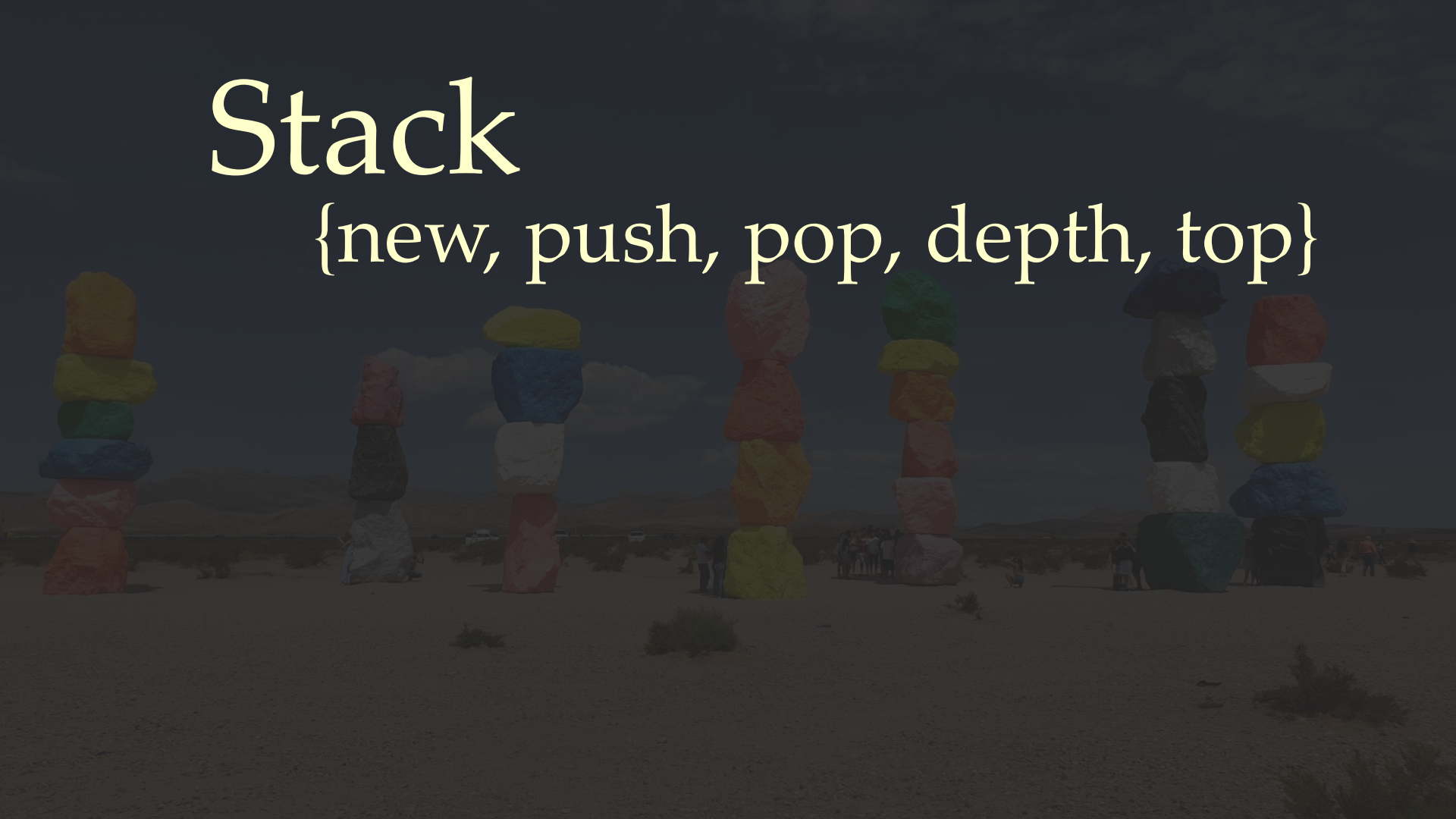

Simple cases
Common cases
Boundary cases
Error cases


Simple cases
Common cases
Boundary cases
Error cases?

```
TEST(A_year_is_supported, if_it_is_positive)
{
    ASSERT_NO_THROW(is_leap_year(std::numeric_limits<int>::
}
TEST(A_year_is_not_supported, if_it_is_0)
{
    ASSERT_THROW(is_leap_year(0), std::invalid_argument);
}
TEST(A_year_is_not_supported, if_it_is_negative)
{
    ASSERT_THROW(is_leap_year(-1), std::invalid_argument);
}
```

```
TEST(A_year_is_supported, if_it_is_positive)
{
    ASSERT_NO_THROW(is_leap_year(INT_MAX));
}
TEST(A_year_is_not_supported, if_it_is_0)
{
    ASSERT_THROW(is_leap_year(0), std::invalid_argument);
}
TEST(A_year_is_not_supported, if_it_is_negative)
{
    ASSERT_THROW(is_leap_year(-1), std::invalid_argument);
}
```

```
TEST(A_year_is_supported, if_it_is_positive)
{
    ASSERT_NO_THROW(is_leap_year(INT_MAX));
}
TEST(A_year_is_not_supported, if_it_is_0)
{
    ASSERT_THROW(is_leap_year(0), std::invalid_argument);
}
TEST(A_year_is_not_supported, if_it_is_negative)
{
    ASSERT_THROW(is_leap_year(-1), std::invalid_argument);
}
```



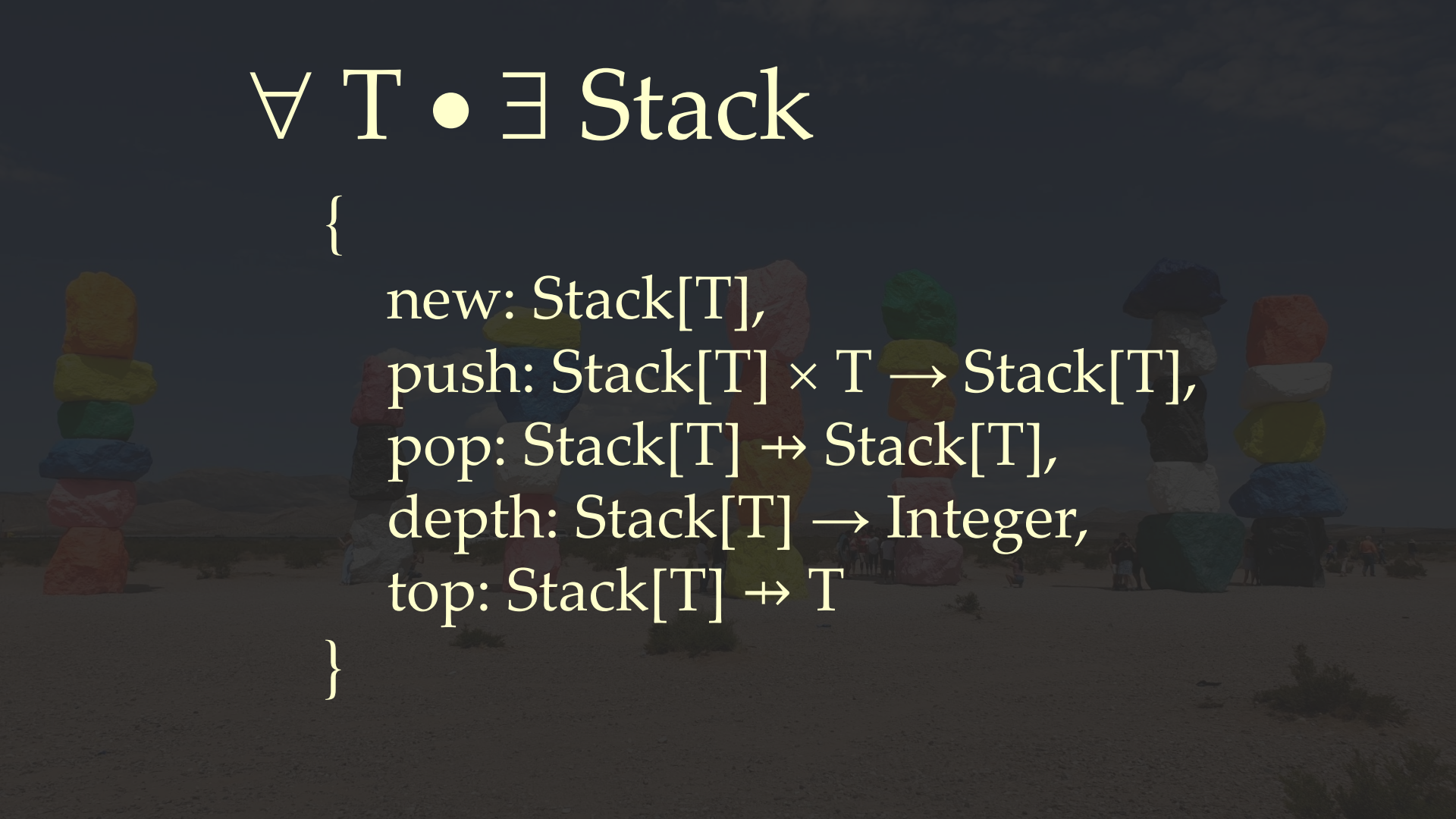
Stack

{new, push, pop, depth, top}

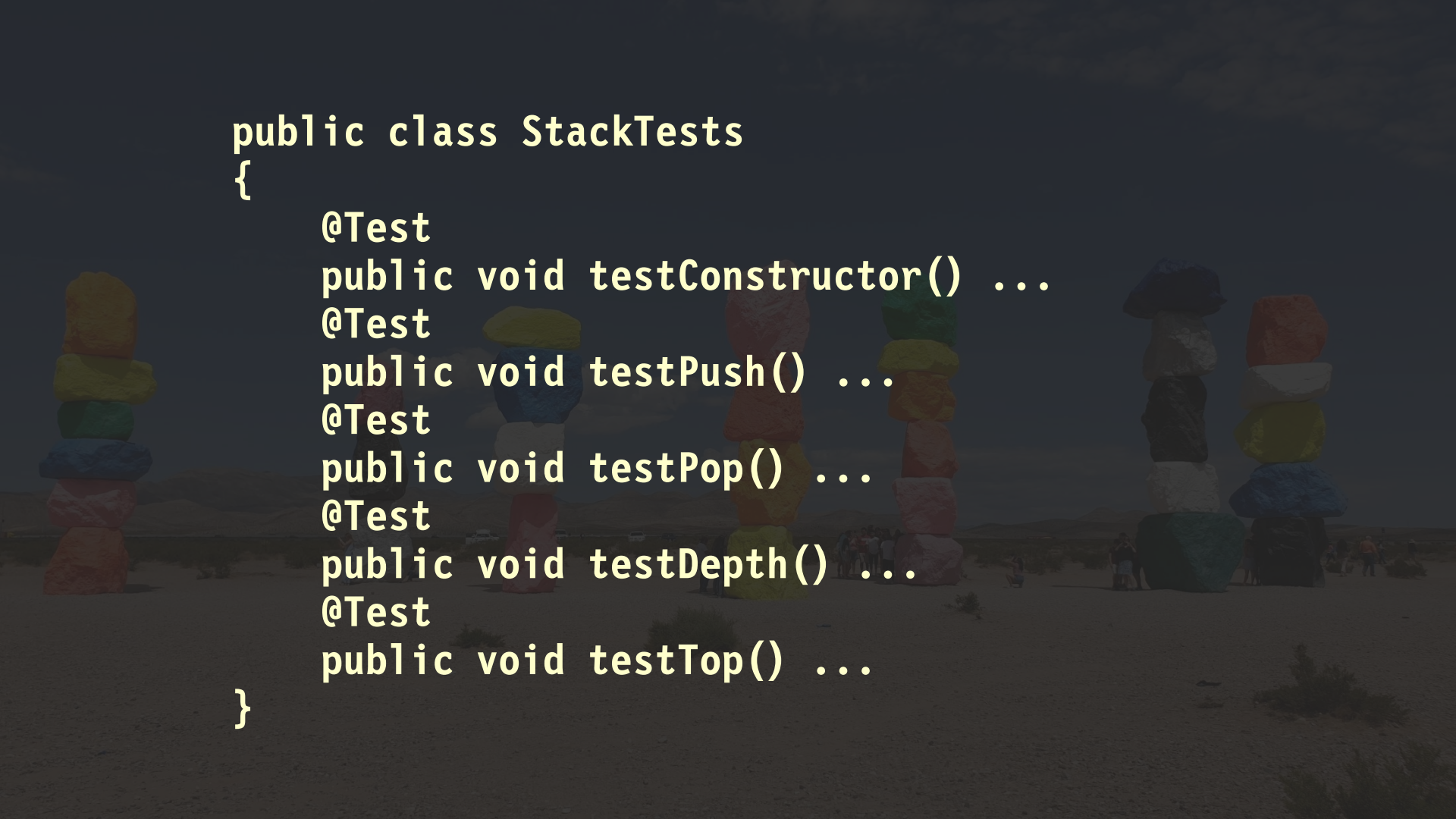
An abstract data type defines a class of abstract objects which is completely characterized by the operations available on those objects.

Barbara Liskov

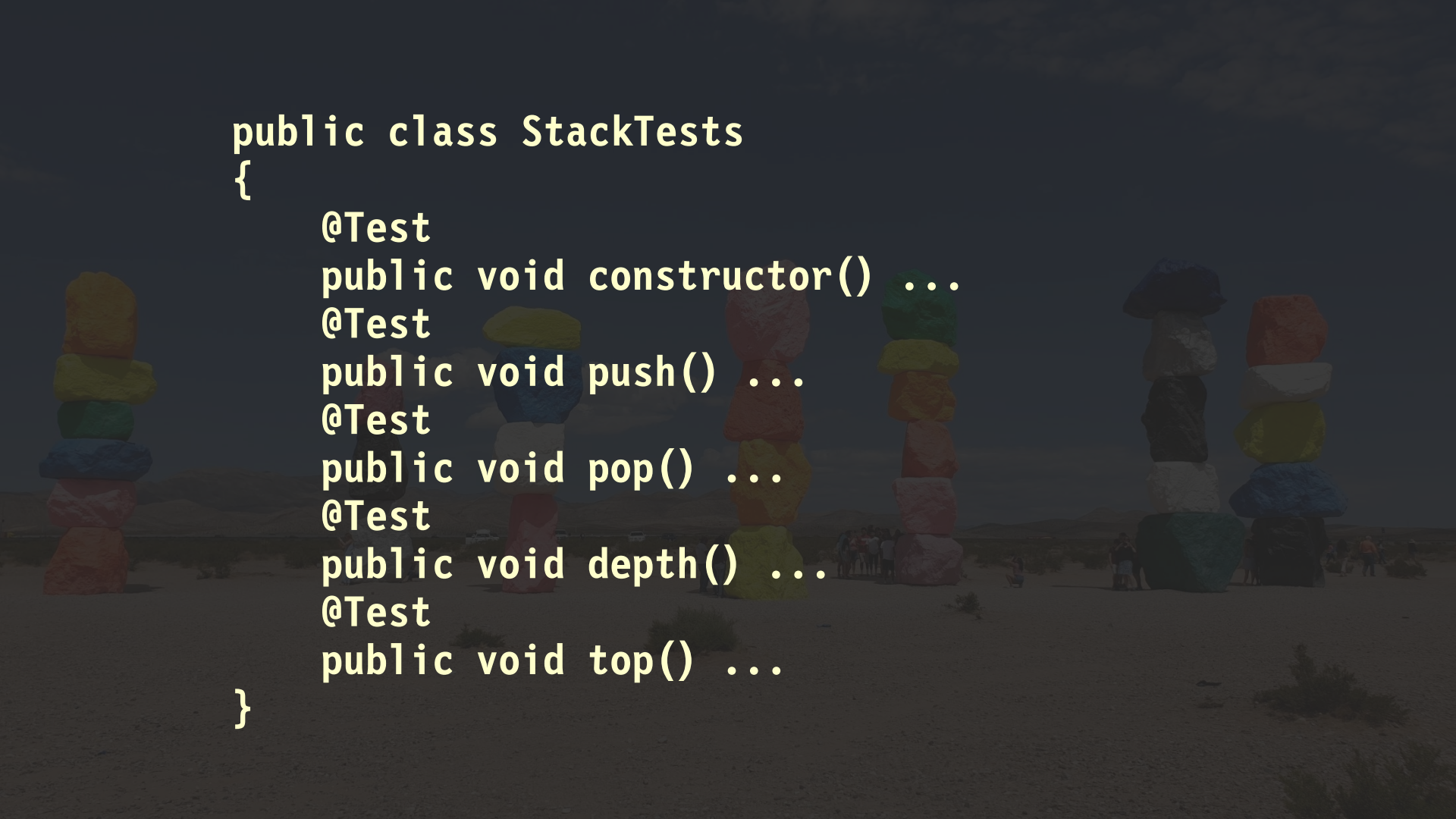
Programming with Abstract Data Types


∀ T • ∃ Stack
{
new: Stack[T],
push: Stack[T] × T → Stack[T],
pop: Stack[T] ⇸ Stack[T],
depth: Stack[T] → Integer,
top: Stack[T] ⇸ T
}

```
public class Stack<T>
{
    ...
    public Stack() ...
    public void push(T newTop) ...
    public void pop() ...
    public int depth() ...
    public T top() ...
}
```

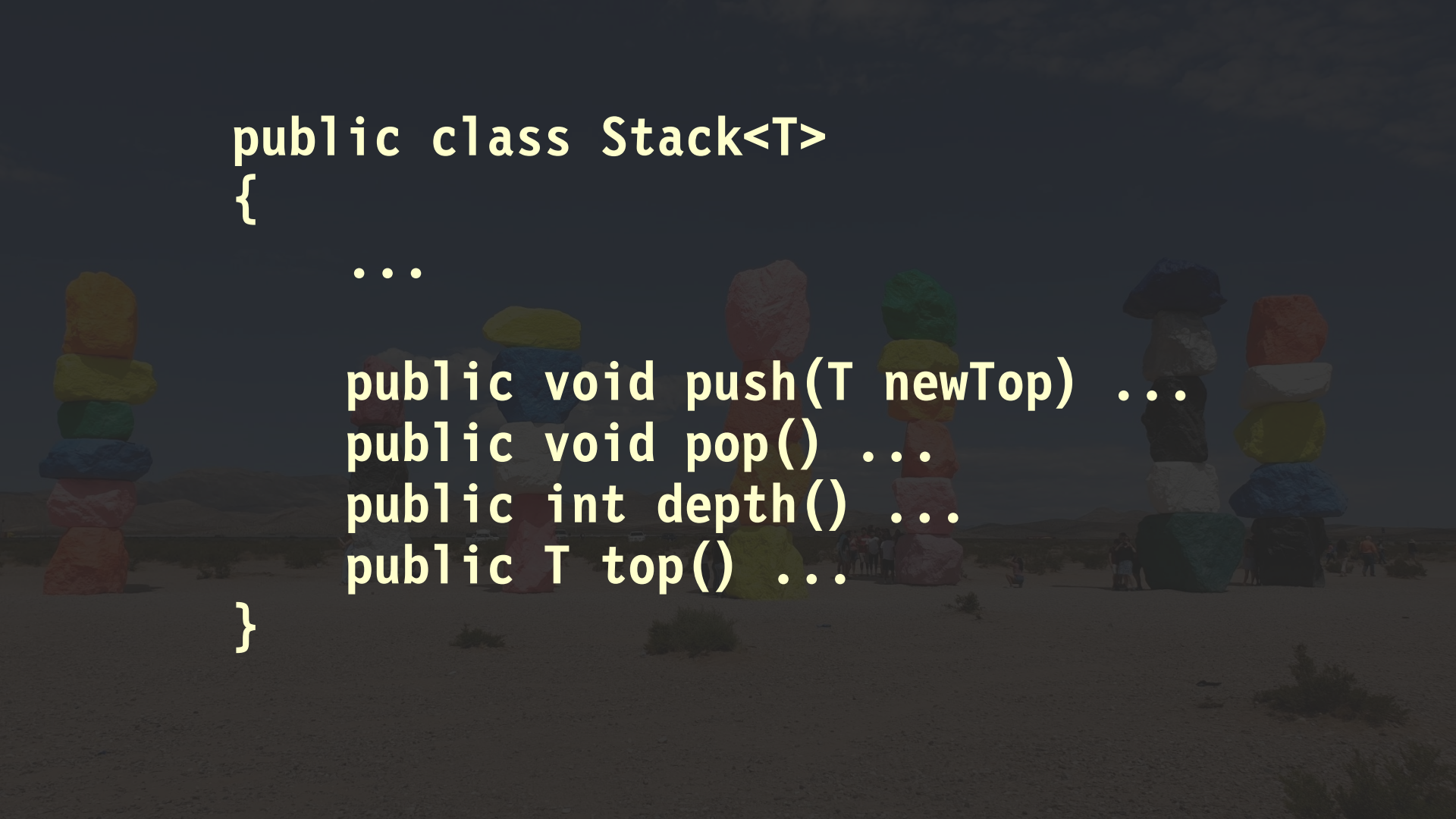


```
public class StackTests
{
    @Test
    public void testConstructor() ...
    @Test
    public void testPush() ...
    @Test
    public void testPop() ...
    @Test
    public void testDepth() ...
    @Test
    public void testTop() ...
}
```

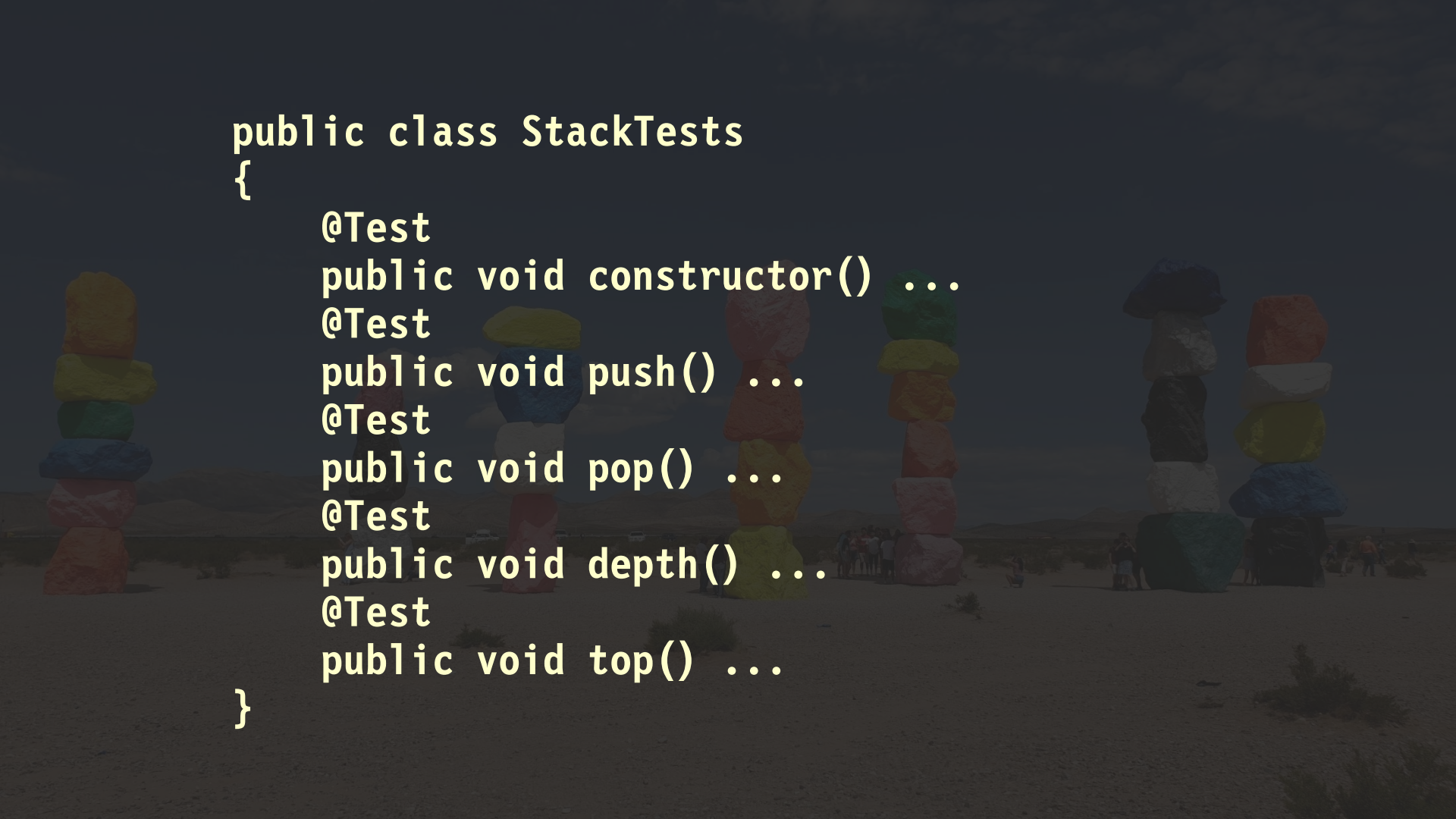


```
public class StackTests
{
    @Test
    public void constructor() ...
    @Test
    public void push() ...
    @Test
    public void pop() ...
    @Test
    public void depth() ...
    @Test
    public void top() ...
}
```

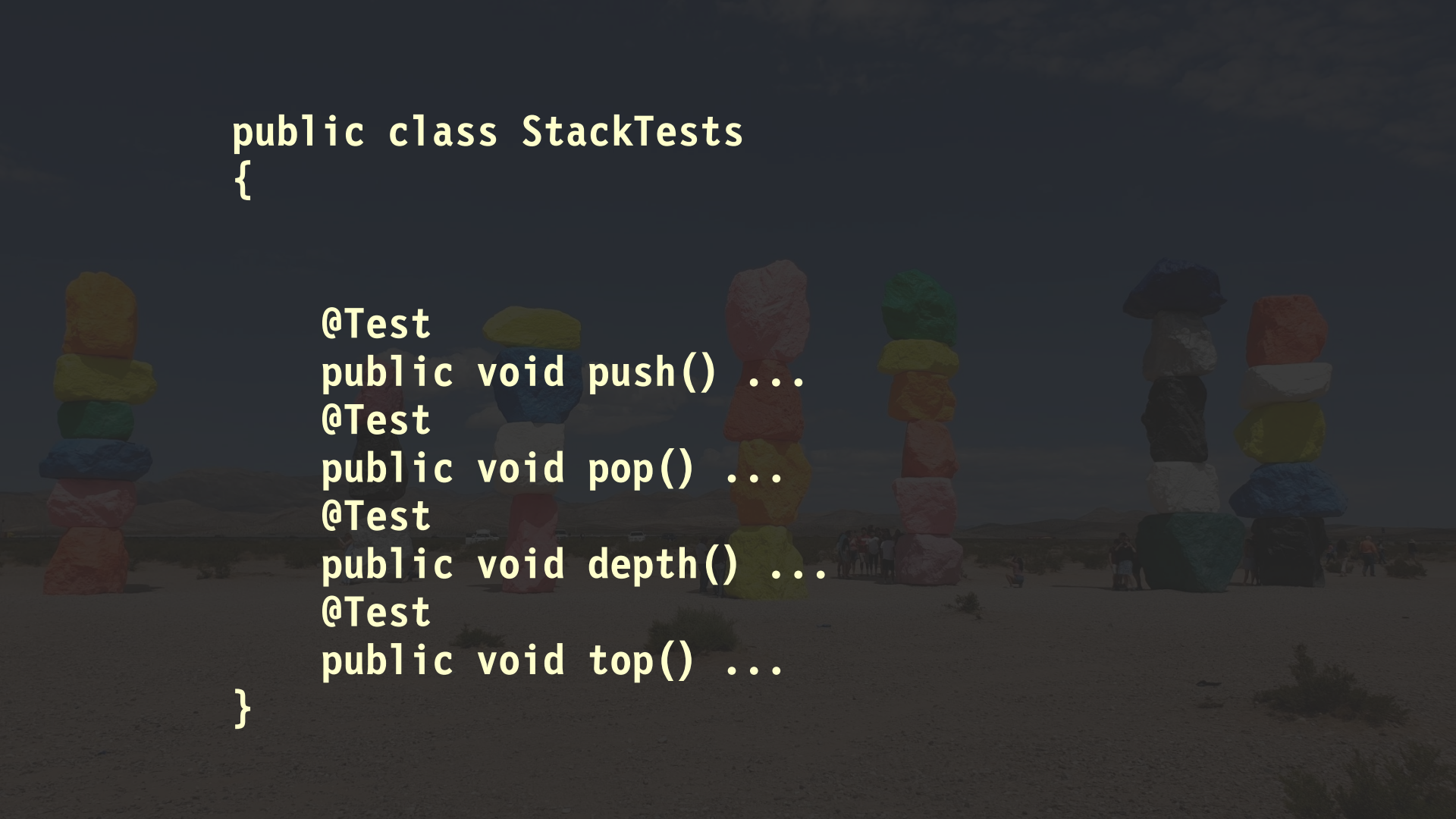
```
public class Stack<T>
{
    ...
    public Stack() ...
    public void push(T newTop) ...
    public void pop() ...
    public int depth() ...
    public T top() ...
}
```



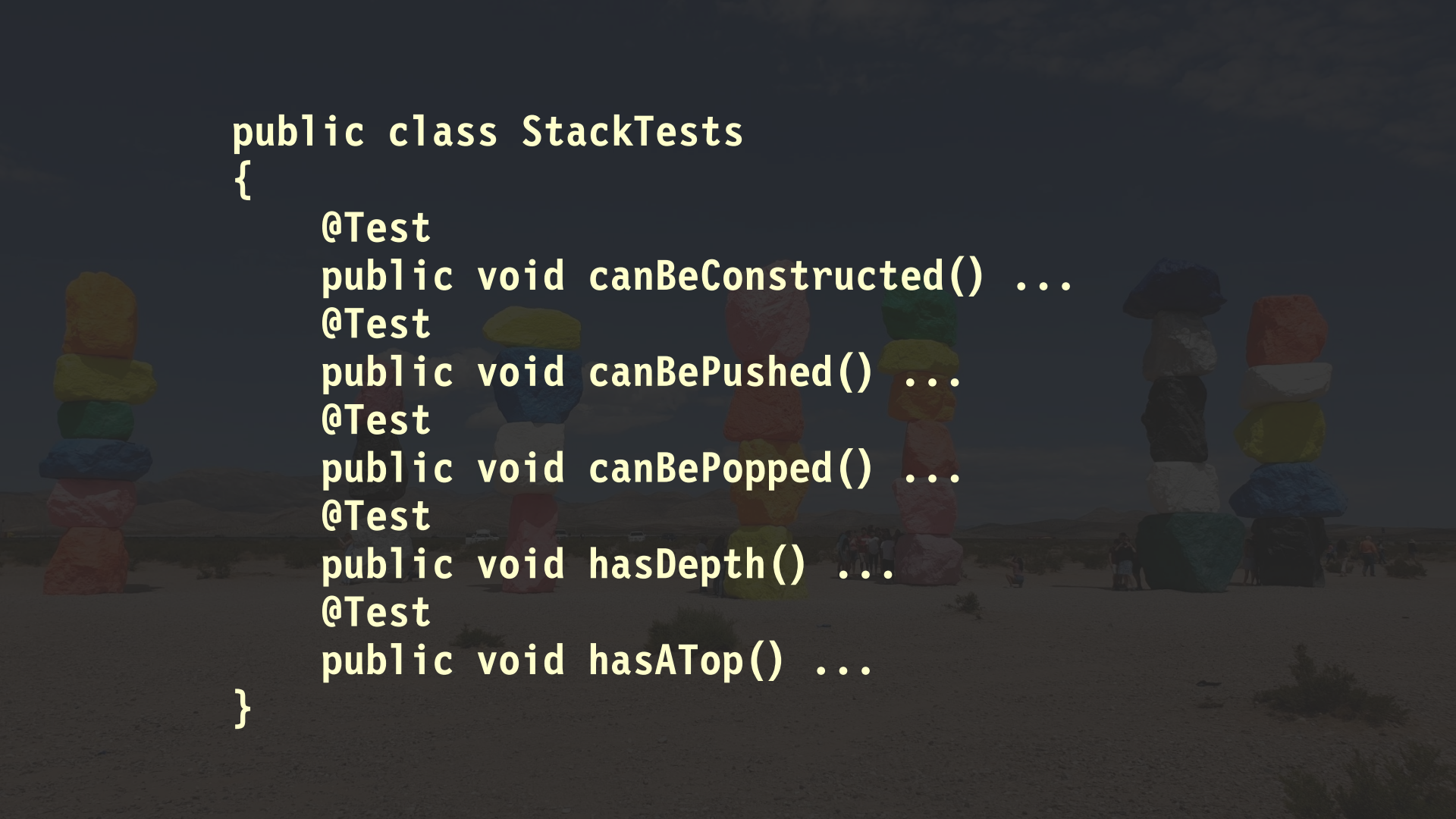
```
public class Stack<T>
{
    ...

    public void push(T newTop) ...
    public void pop() ...
    public int depth() ...
    public T top() ...
}
```



```
public class StackTests
{
    @Test
    public void constructor() ...
    @Test
    public void push() ...
    @Test
    public void pop() ...
    @Test
    public void depth() ...
    @Test
    public void top() ...
}
```

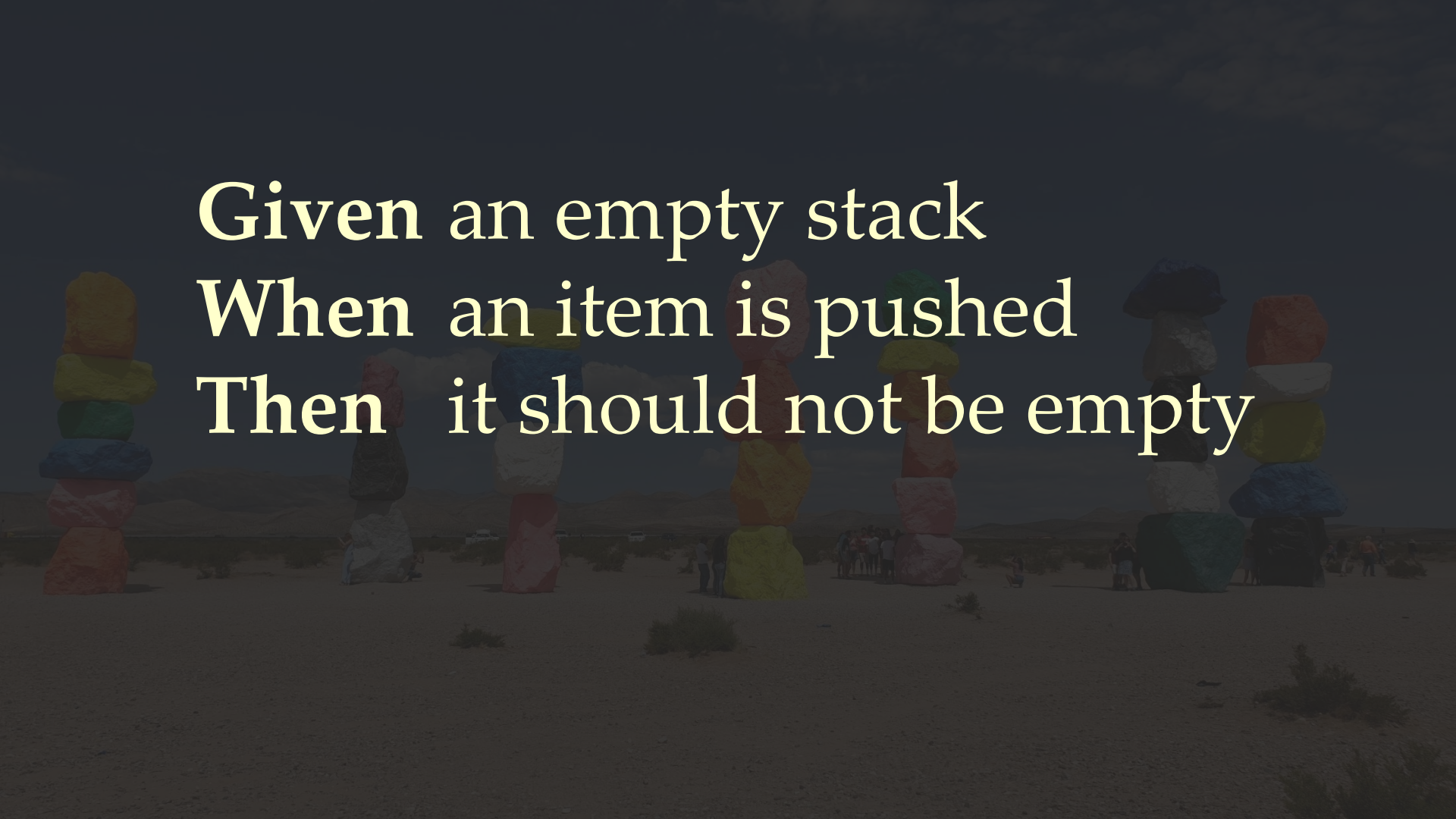


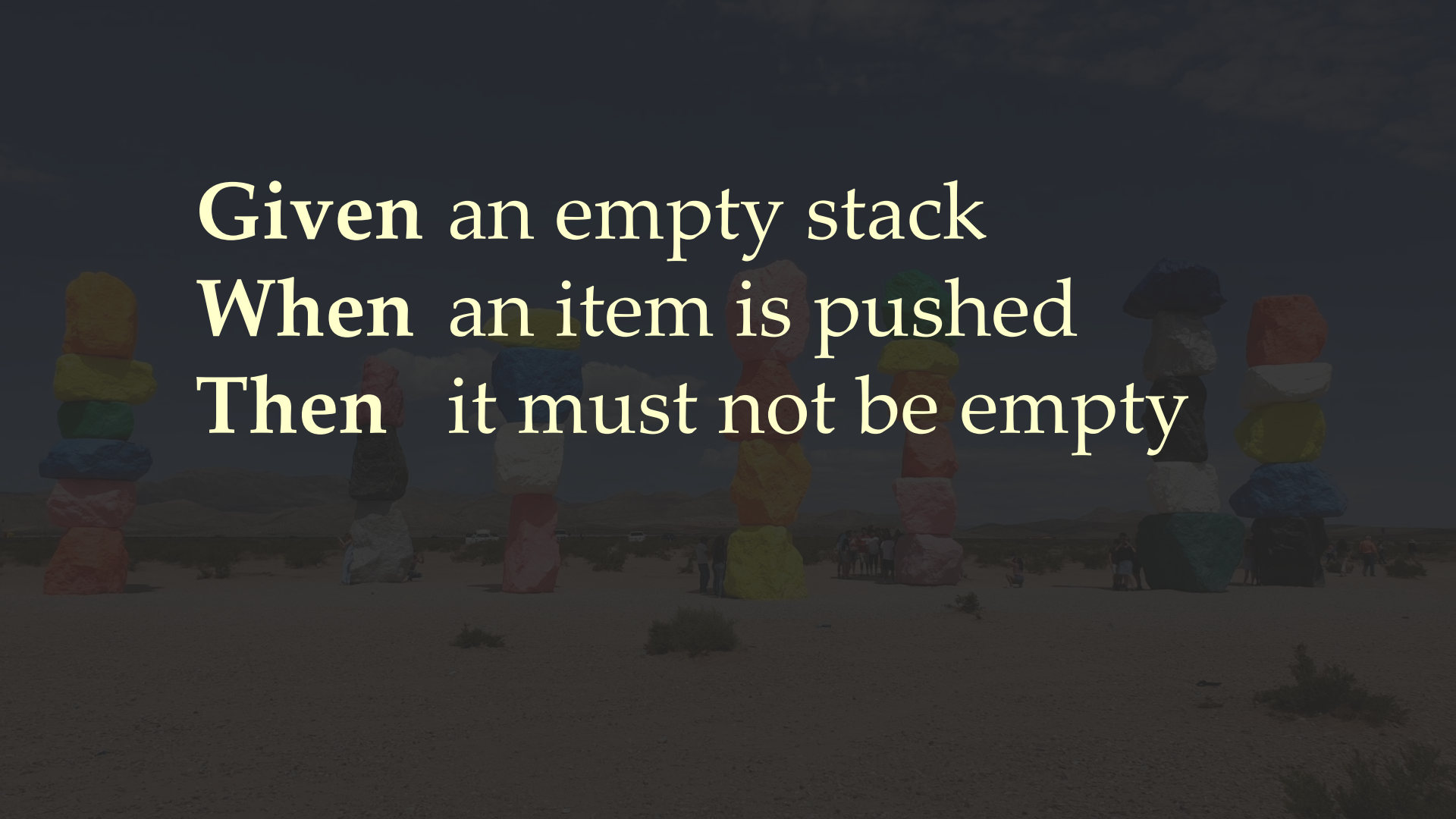
```
public class StackTests
{


    @Test
    public void push() ...
    @Test
    public void pop() ...
    @Test
    public void depth() ...
    @Test
    public void top() ...
}
```

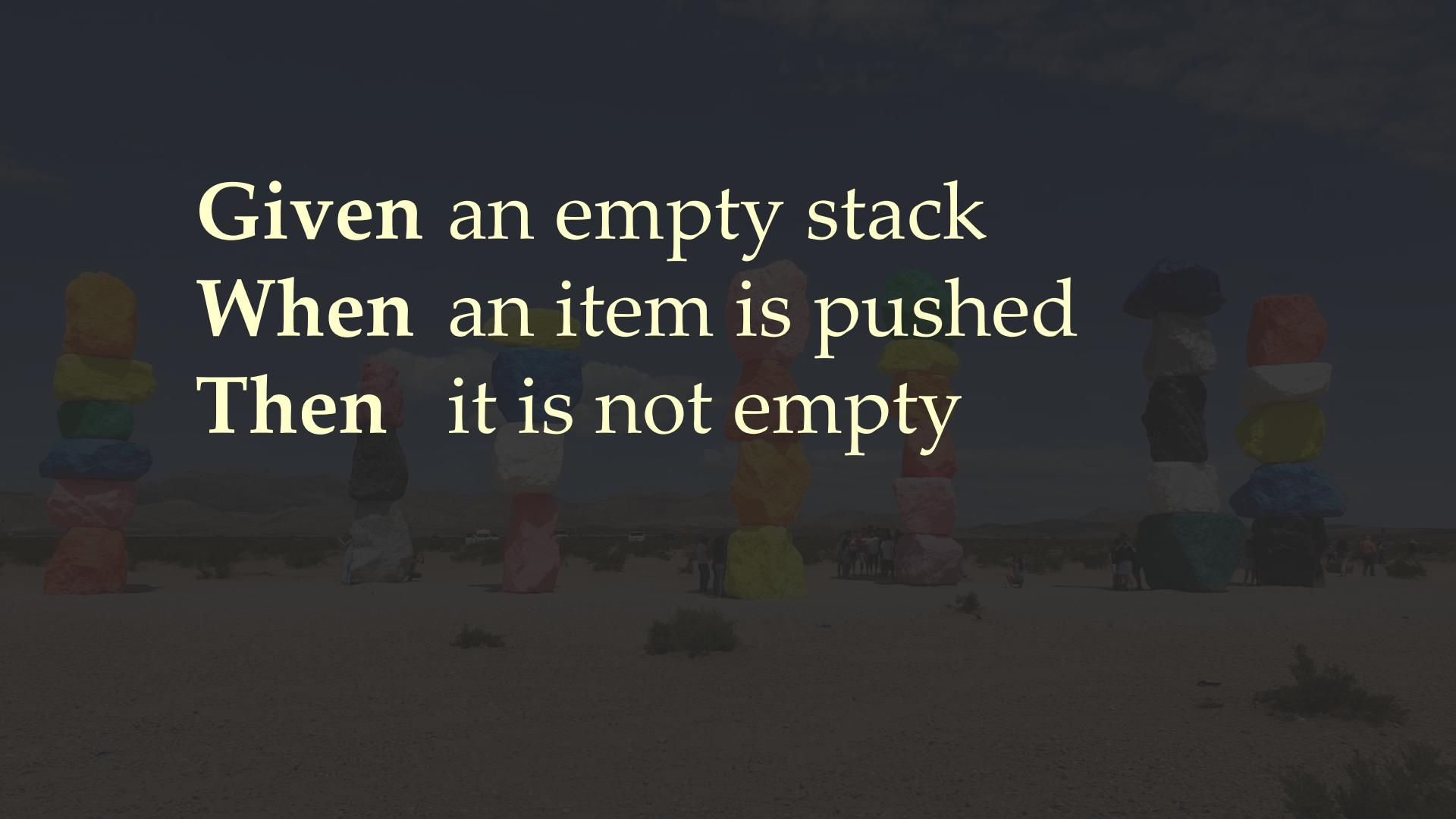


```
public class StackTests
{
    @Test
    public void canBeConstructed() ...
    @Test
    public void canBePushed() ...
    @Test
    public void canBePopped() ...
    @Test
    public void hasDepth() ...
    @Test
    public void hasATop() ...
}
```

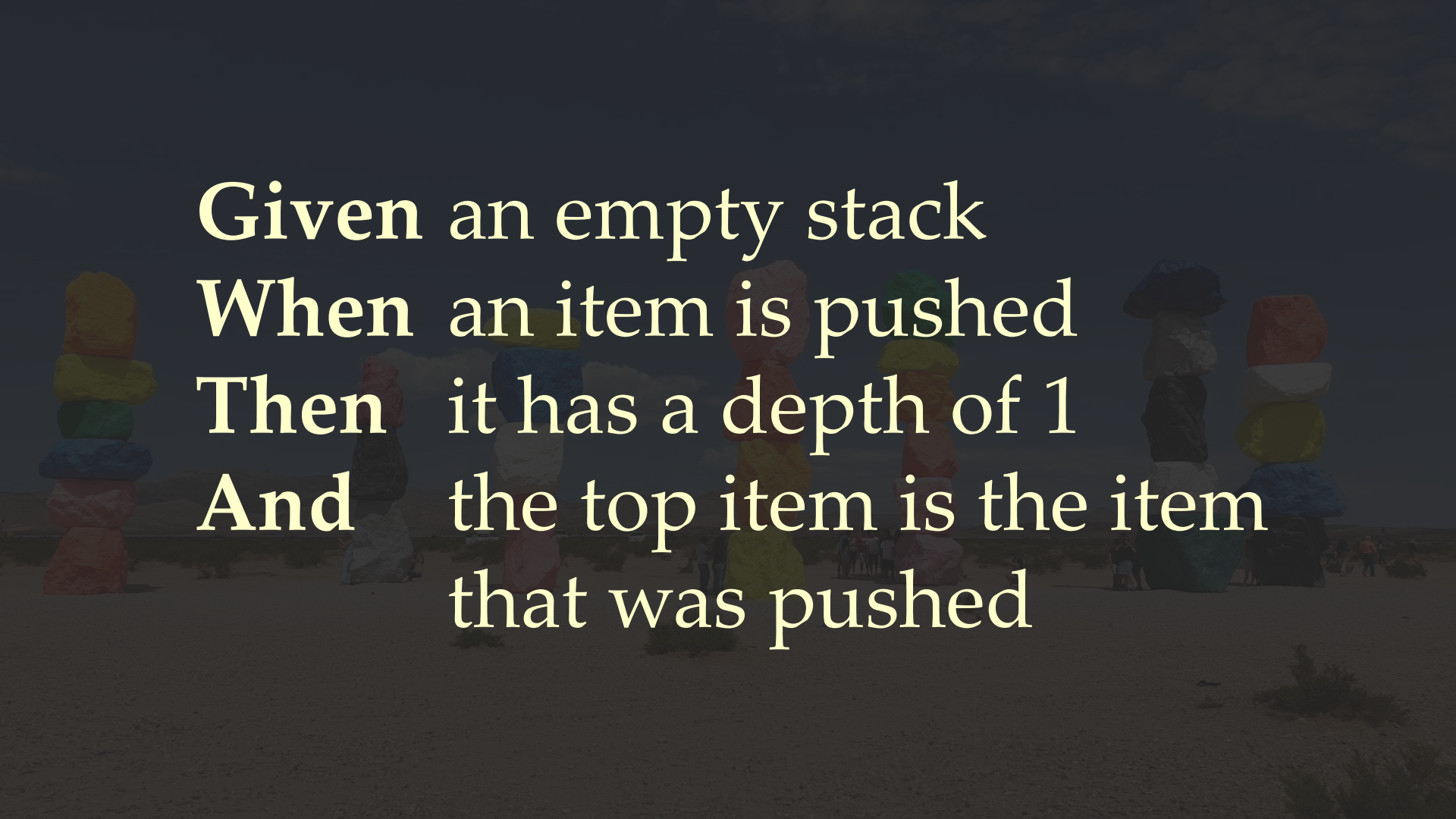
```
public class StackTests
{
    @Test
    public void canBeConstructed() ...
    @Test
    public void canBePushed() ...
    @Test
    public void canSometimesBePopped() ...
    @Test
    public void hasDepth() ...
    @Test
    public void sometimesHasATop() ...
}
```


Given an empty stack
When an item is pushed
Then it should not be empty

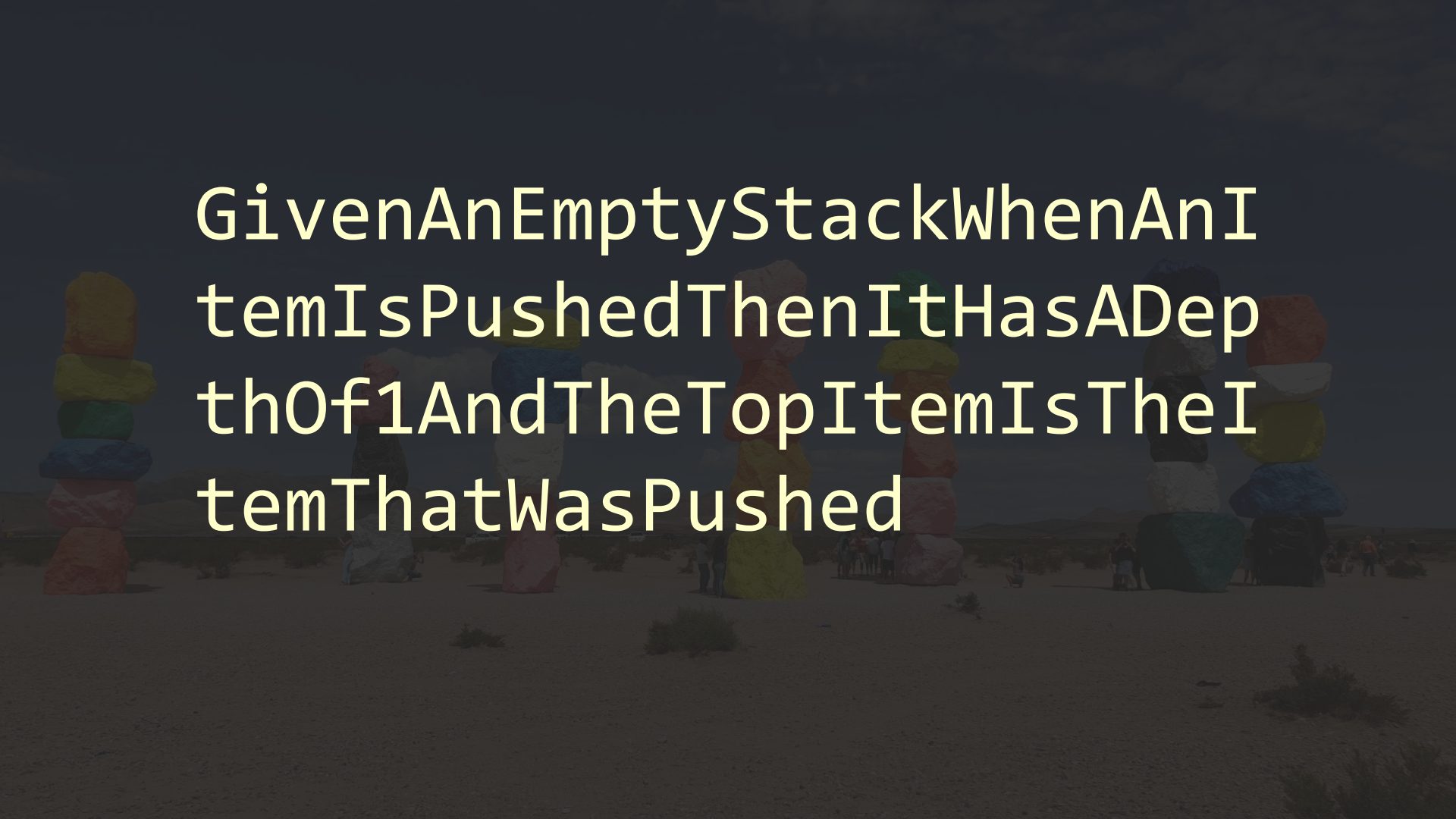

Given an empty stack
When an item is pushed
Then it must not be empty

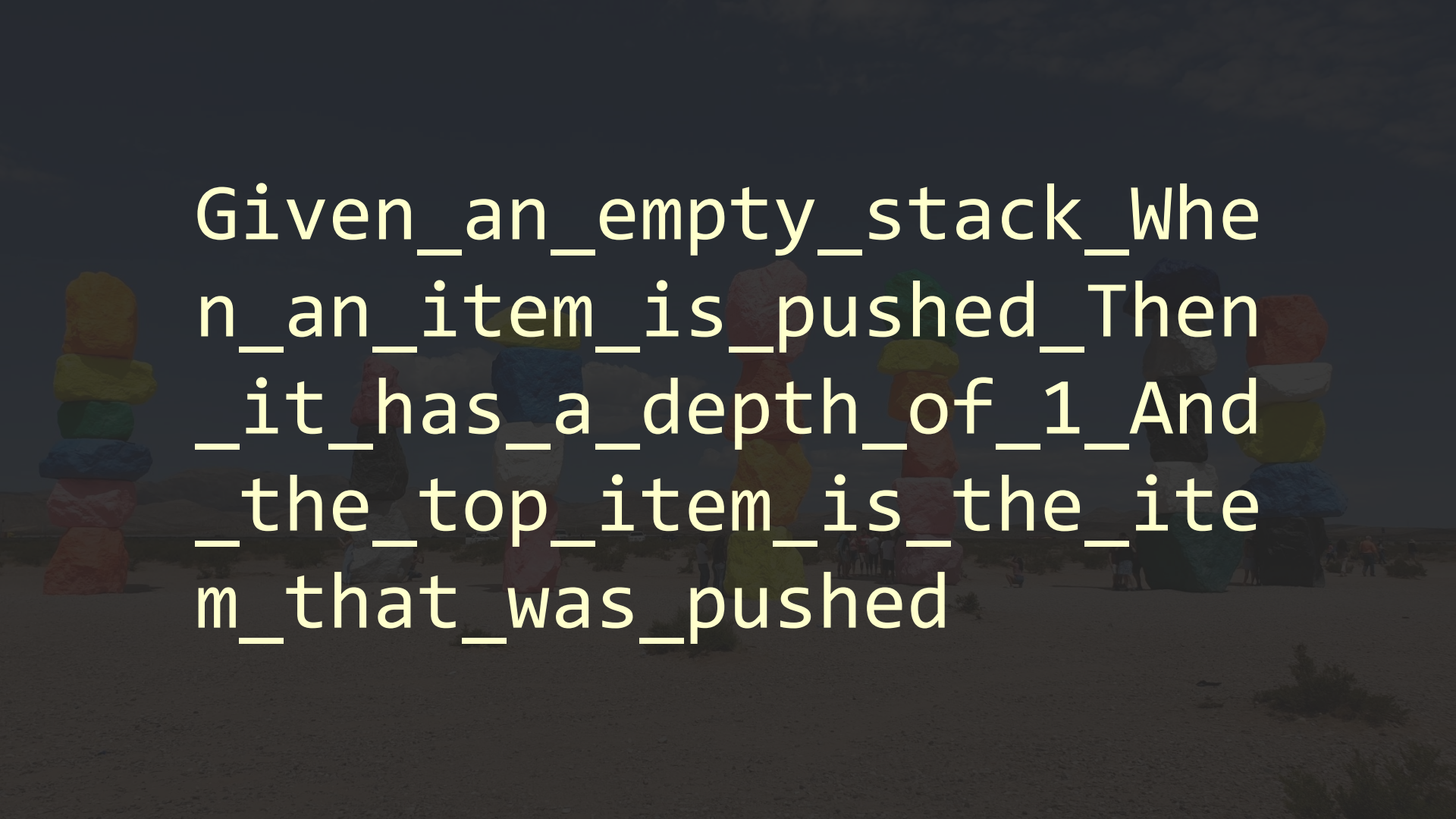


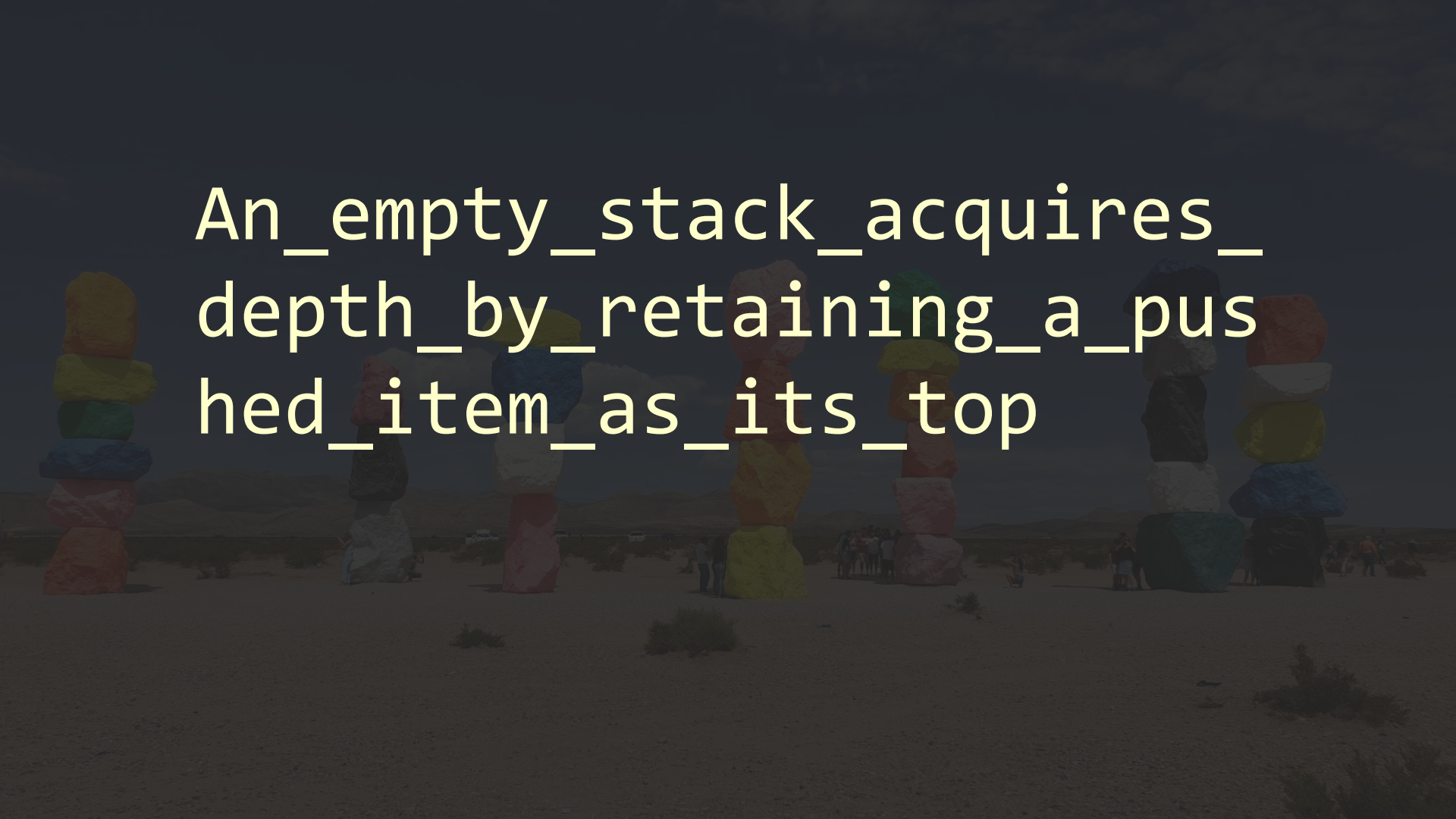
Given an empty stack
When an item is pushed
Then it is not empty



Given an empty stack
When an item is pushed
Then it has a depth of 1
And the top item is the item that was pushed


GivenAnEmptyStackWhenAnItemIsPushedThenItHasADepthOf1AndTheTopItemIsTheItemThatWasPushed

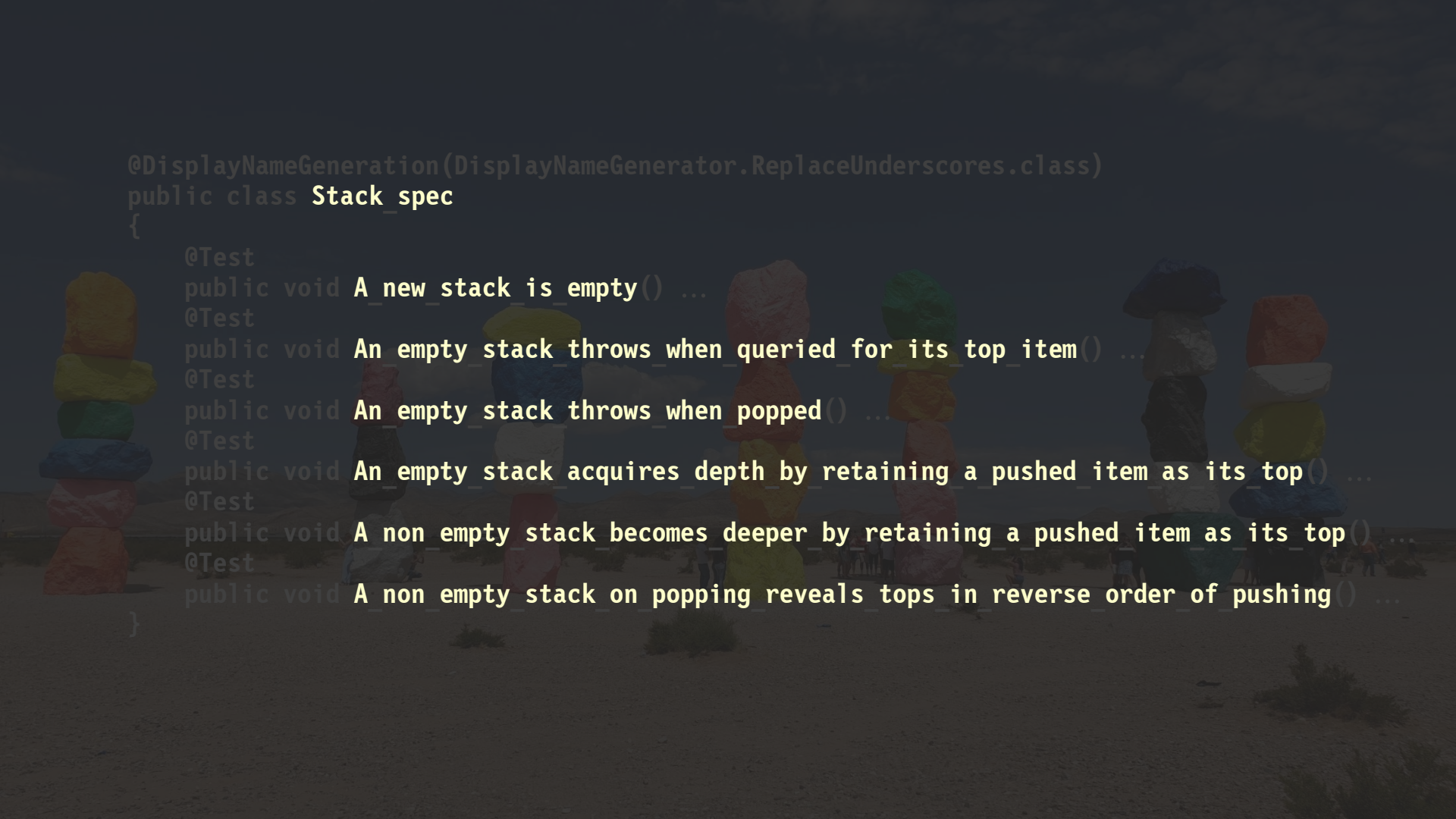

Given_an_empty_stack_When_an_item_is_pushed_Then_it_has_a_depth_of_1_And_the_top_item_is_the_item_that_was_pushed


An_empty_stack_acquires_depth_by_retaining_a_pushed_item_as_its_top

```
public class Stack_spec
{
    ...
    @Test
    public void An_empty_stack_acquires_depth_by_retaining_a_pushed_item_as_its_top()
    {
        Stack<String> stack = new Stack<>();

        stack.push("rock");

        assertEquals(1, stack.depth());
        assertEquals("rock", stack.top());
    }
    ...
}
```

```
public class Stack_spec
{
    @Test
    public void A_new_stack_is_empty() ...
    @Test
    public void An_empty_stack_throws_when_queried_for_its_top_item() ...
    @Test
    public void An_empty_stack_throws_when_popped() ...
    @Test
    public void An_empty_stack_acquires_depth_by_retaining_a_pushed_item_as_its_top() ...
    @Test
    public void A_non_empty_stack_becomes_deeper_by_retaining_a_pushed_item_as_its_top() ...
    @Test
    public void A_non_empty_stack_on_popping_reveals_tops_in_reverse_order_of_pushing() ...
}
```

```
@DisplayNameGeneration(DisplayNameGenerator.ReplaceUnderscores.class)
public class Stack_spec
{
    @Test
    public void A_new_stack_is_empty() ...
    @Test
    public void An_empty_stack_throws_when_queried_for_its_top_item() ...
    @Test
    public void An_empty_stack_throws_when_popped() ...
    @Test
    public void An_empty_stack_acquires_depth_by_retaining_a_pushed_item_as_its_top() ...
    @Test
    public void A_non_empty_stack_becomes_deeper_by_retaining_a_pushed_item_as_its_top() ...
    @Test
    public void A_non_empty_stack_on_popping_reveals_tops_in_reverse_order_of_pushing() ...
}
```


@DisplayNameGeneration(DisplayNameGenerator.ReplaceUnderscores.class)
public class Stack_spec
{
@Test
public void A_new_stack_is_empty() ...
@Test
public void An_empty_stack_throws_when_queried_for_its_top_item() ...
@Test
public void An_empty_stack_throws_when_popped() ...
@Test
public void An_empty_stack_acquires_depth_by_retaining_a_pushed_item_as_its_top() ...
@Test
public void A_non_empty_stack_becomes_deeper_by_retaining_a_pushed_item_as_its_top() ...
@Test
public void A_non_empty_stack_on_popping_reveals_tops_in_reverse_order_of_pushing() ...
}

```
public class Stack_spec
{
    ...
    @Test
    public void An_empty_stack_acquires_depth_by_retaining_a_pushed_item_as_its_top()
    {
        Stack<String> stack = new Stack<>();

        stack.push("rock");

        assertEquals(1, stack.depth());
        assertEquals("rock", stack.top());
    }
    ...
}
```


Kevlin Henney 编
李军 译 吕骏 审校
電子工業出版社
O'REILLY
オライリー・ジャパン
97
Collective Wisdom
from the Experts
97 Things Every
Programmer
Should Know
O'REILLY
Edited by Kevlin Henney

Write Tests for People

Gerard Meszaros

For each usage scenario, the test(s):

- Describe the context, starting point, or preconditions that must be satisfied
- Illustrate how the software is invoked
- Describe the expected results or postconditions to be verified

Gerard Meszaros

```
public class Stack_spec
{
    ...
    @Test
    public void An_empty_stack_acquires_depth_by_retaining_a_pushed_item_as_its_top()
    {
        // Arrange:
        Stack<String> stack = new Stack<>();
        // Act:
        stack.push("rock");
        // Assert:
        assertEquals(1, stack.depth());
        assertEquals("rock", stack.top());
    }
    ...
}
```

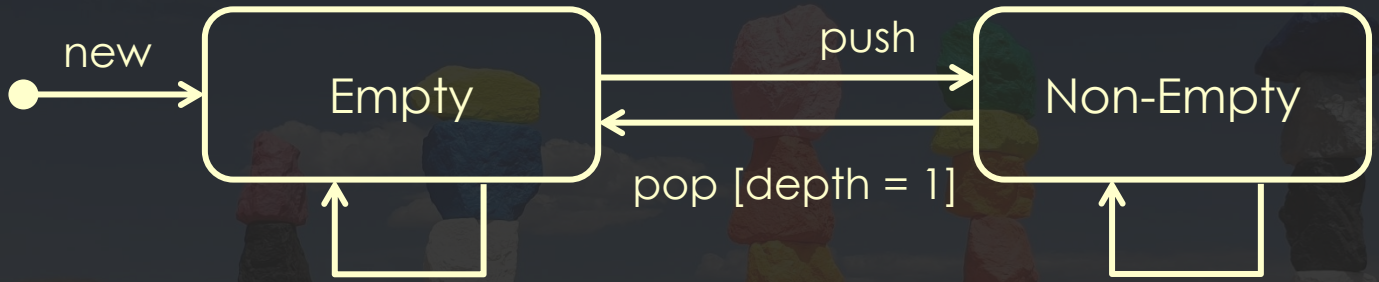
```
public class Stack_spec
{
    ...
    @Test
    public void An_empty_stack_acquires_depth_by_retaining_a_pushed_item_as_its_top()
    {
        // Arrange:
        Stack<String> stack = new Stack<>();
        // Act:
        stack.push("rock");
        // Assert:
        assertEquals(1, stack.depth());
        assertEquals("rock", stack.top());
    }
    ...
}
```

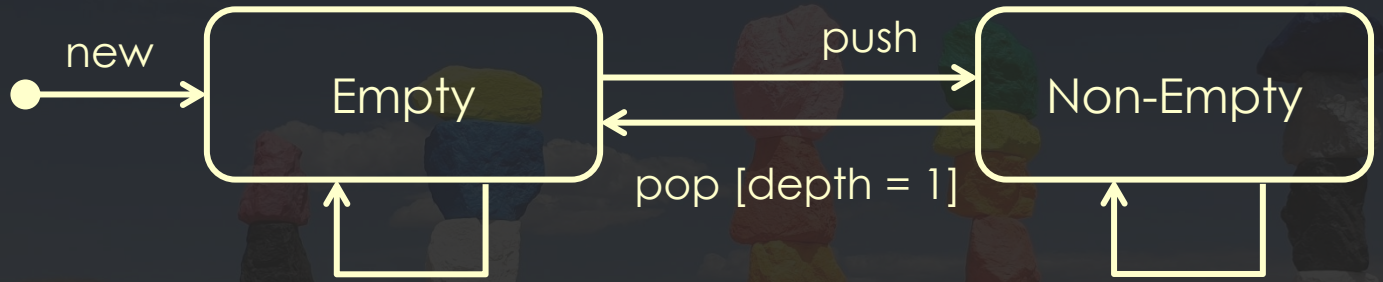
```
public class Stack_spec
{
    ...
    @Test
    public void An_empty_stack_acquires_depth_by_retaining_a_pushed_item_as_its_top()
    {
        // Given:
        Stack<String> stack = new Stack<>();
        // When:
        stack.push("rock");
        // Then:
        assertEquals(1, stack.depth());
        assertEquals("rock", stack.top());
    }
    ...
}
```

Thinking in States

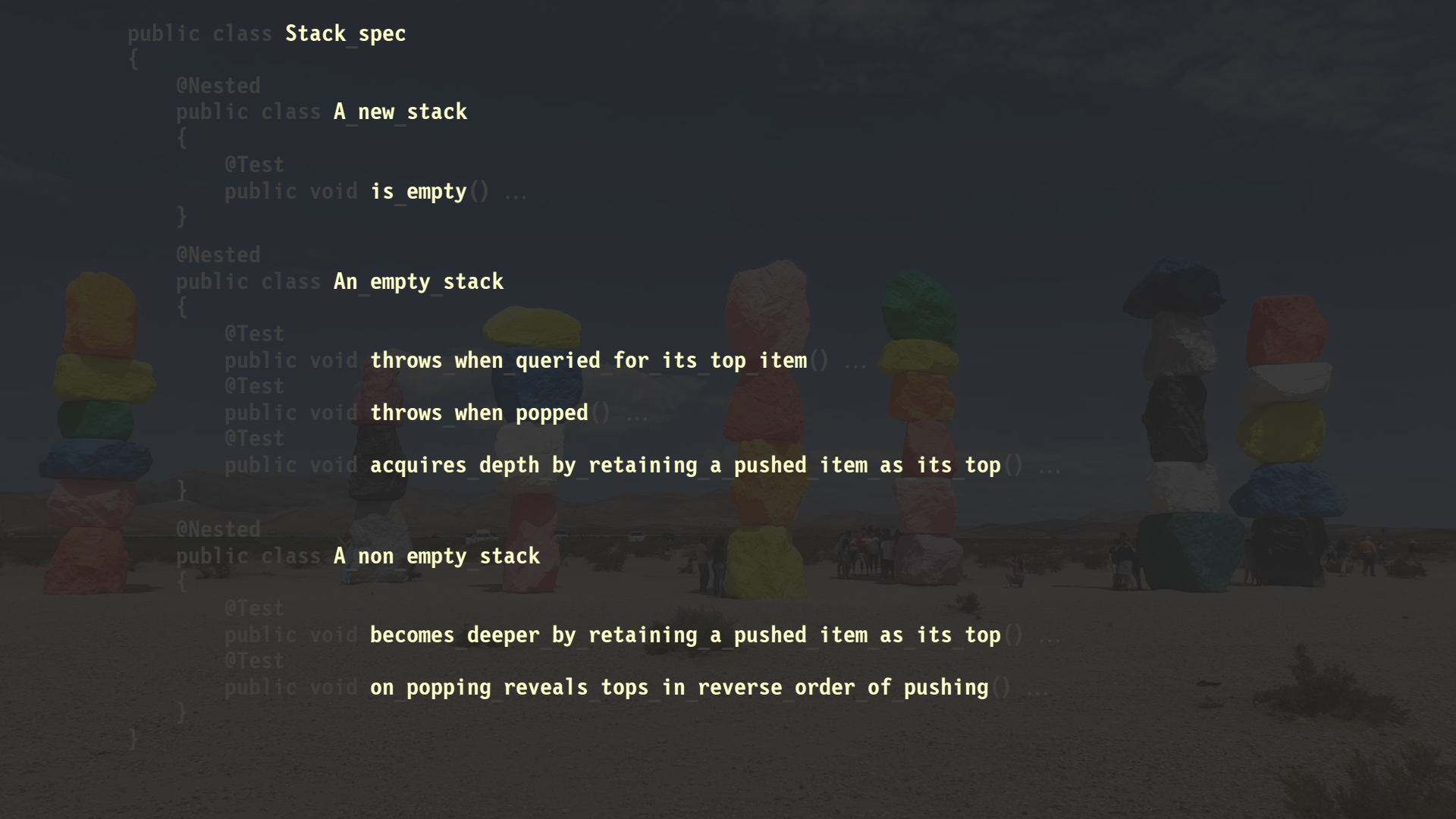
In most real-world situations, people's relaxed attitude to state is not an issue. Unfortunately, however, many programmers are quite vague about state too — and that is a problem.

Niclas Nilsson


new
Empty
push
Non-Empty
pop [depth = 1]
depth
depth
top
push
pop [depth > 1]

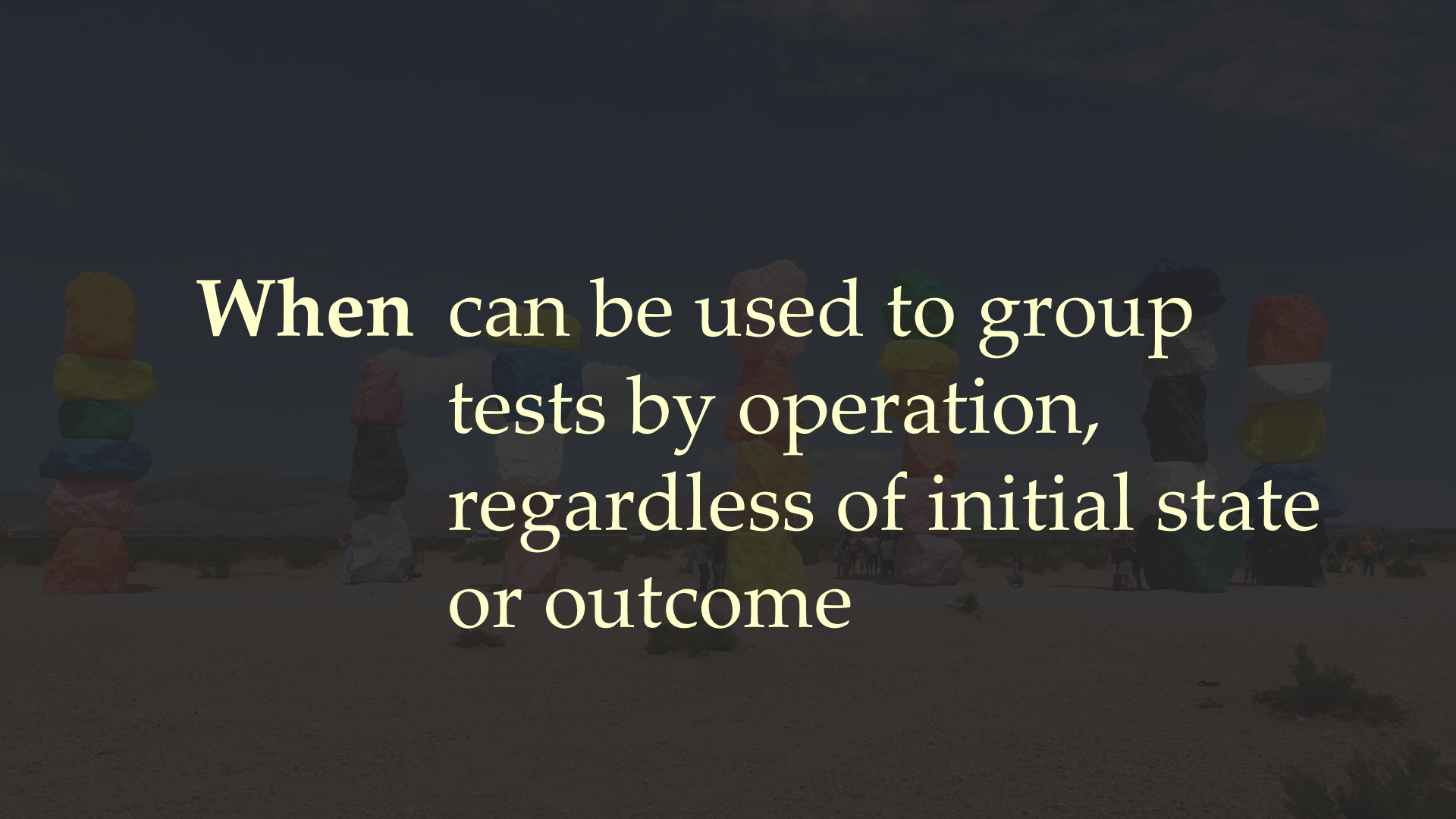

new
Empty
push
Non-Empty
pop [depth = 1]
depth
top / throw
pop / throw
depth
top
push
pop [depth > 1]

```
public class Stack_spec
{
    @Nested
    public class A_new_stack
    {
        @Test
        public void is_empty() ...
    }

    @Nested
    public class An_empty_stack
    {
        @Test
        public void throws_when_queried_for_its_top_item() ...
        @Test
        public void throws_when_popped() ...
        @Test
        public void acquires_depth_by_retaining_a_pushed_item_as_its_top() ...
    }

    @Nested
    public class A_non_empty_stack
    {
        @Test
        public void becomes_deeper_by_retaining_a_pushed_item_as_its_top() ...
        @Test
        public void on_popping_reveals_tops_in_reverse_order_of_pushing() ...
    }
}
```

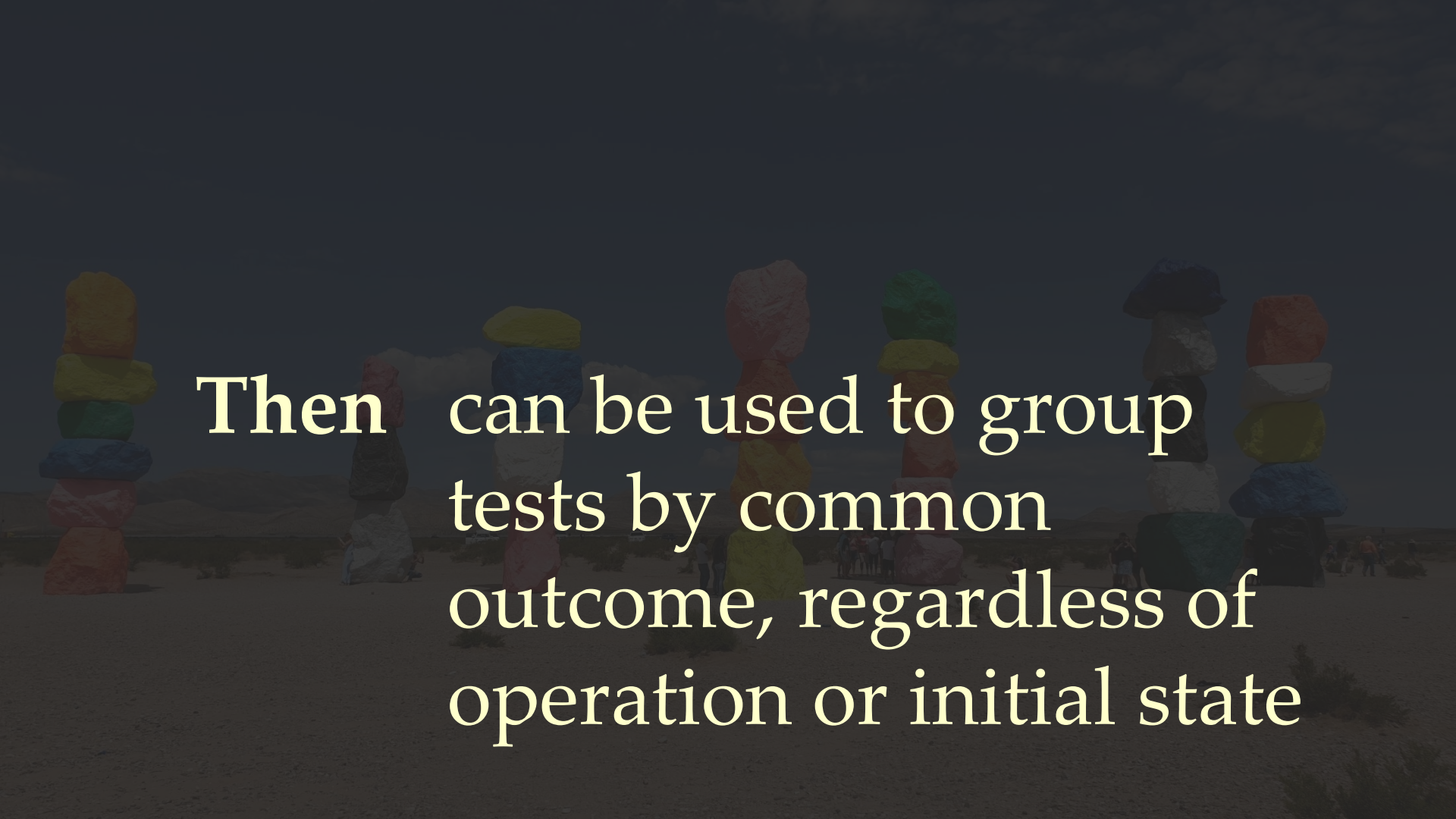


```
public class Stack_spec
{
    @Nested
    public class A_new_stack
    {
        @Test
        public void is_empty() ...
    }

    @Nested
    public class An_empty_stack
    {
        @Test
        public void throws_when_queried_for_its_top_item() ...
        @Test
        public void throws_when_popped() ...
        @Test
        public void acquires_depth_by_retaining_a_pushed_item_as_its_top() ...
    }

    @Nested
    public class A_non_empty_stack
    {
        @Test
        public void becomes_deeper_by_retaining_a_pushed_item_as_its_top() ...
        @Test
        public void on_popping_reveals_tops_in_reverse_order_of_pushing() ...
    }
}
```

Given can be used to group tests for operations with respect to common initial state



When can be used to group tests by operation, regardless of initial state or outcome

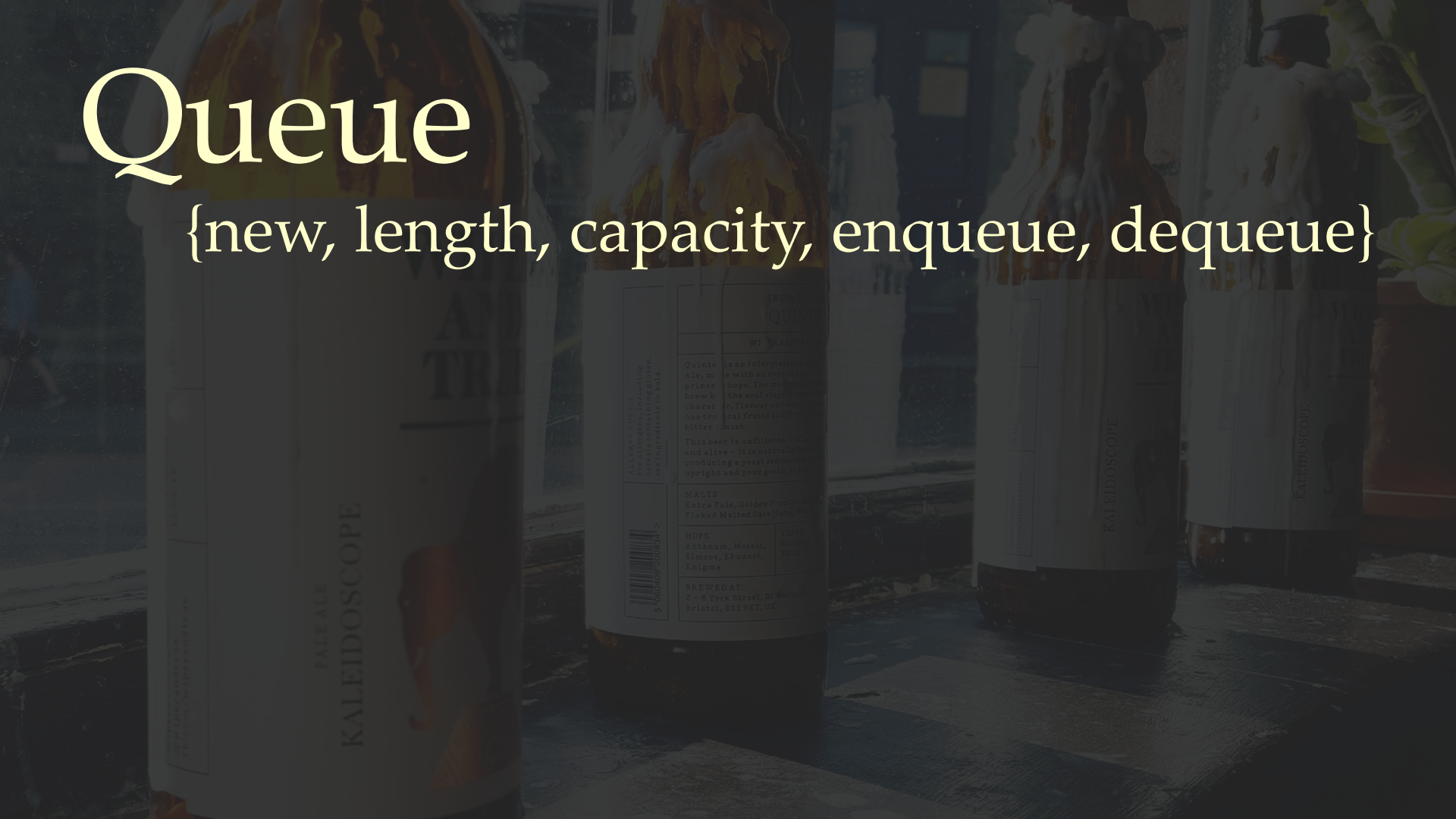


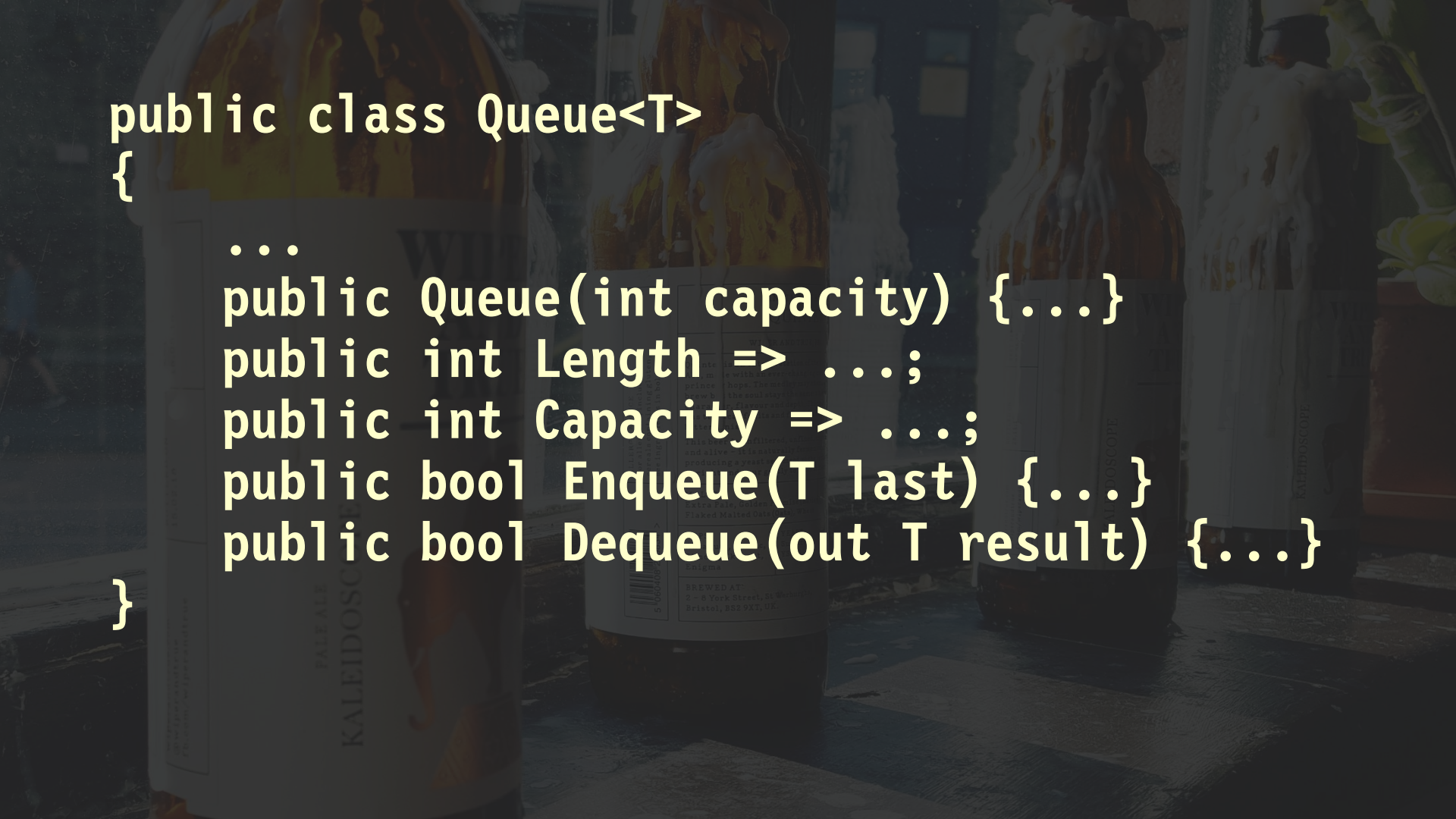
Then can be used to group tests by common outcome, regardless of operation or initial state


PALE ALE
KALEIDOSCOPE
KALEIDOSCOPE

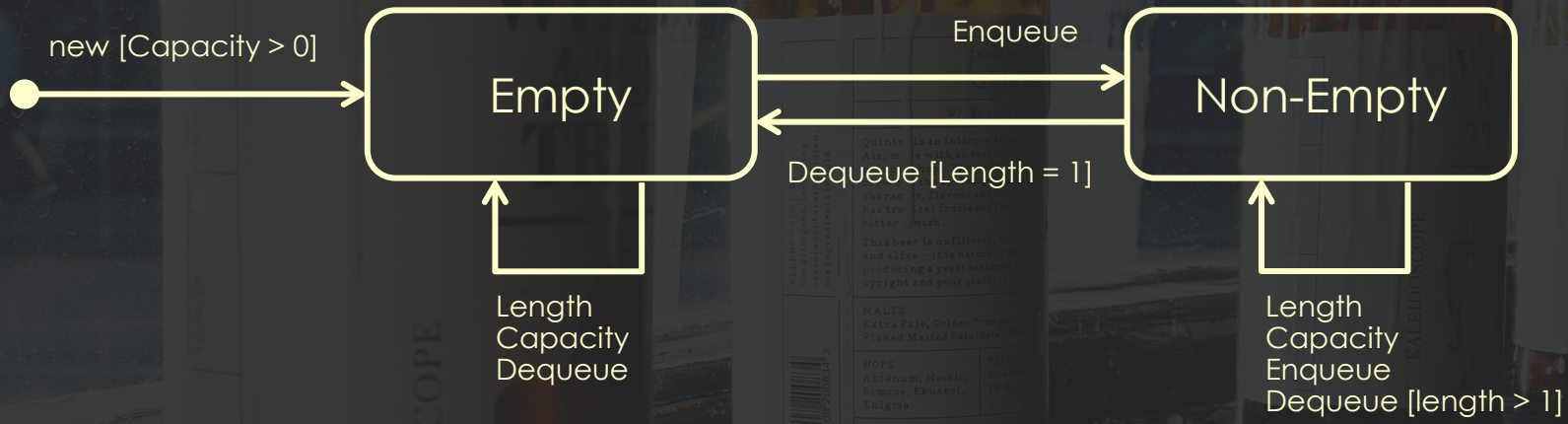
Queue

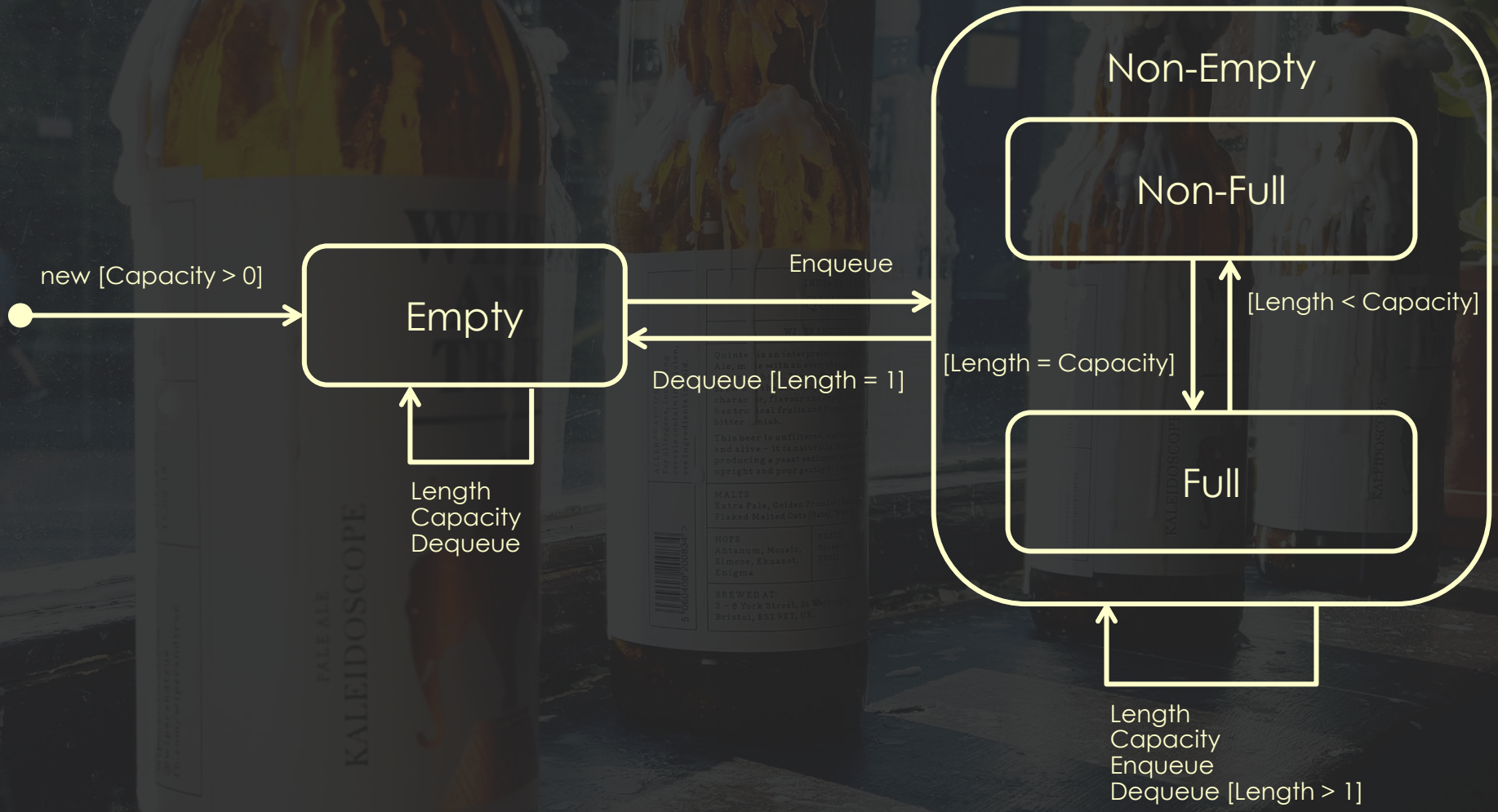
{new, length, capacity, enqueue, dequeue}





```
public class Queue<T>
{
    ...
    public Queue(int capacity) {...}
    public int Length => ...;
    public int Capacity => ...;
    public bool Enqueue(T last) {...}
    public bool Dequeue(out T result) {...}
}
```


new [Capacity > 0]
Empty
Enqueue
Non-Empty
Dequeue [Length = 1]
Length
Capacity
Dequeue
Length
Capacity
Enqueue
Dequeue [length > 1]


Non-Empty
Non-Full
Empty
Full
new [Capacity > 0]
Enqueue
Dequeue [Length = 1]
[Length = Capacity]
[Length < Capacity]
Length
Capacity
Dequeue
Length
Capacity
Enqueue
Dequeue [Length > 1]

```
namespace Queue_spec ...
    public class A_new_queue ...
        public void is_empty() ...
        public void preserves_positive_bounding_capacity() ...
        public void rejects_a_zero_bounding_capacity() ...
        public void rejects_a_negative_bounding_capacity() ...
    public class An_empty_queue ...
        public void dequeues_default_values() ...
        public void becomes_non_empty_when_value_enqueued() ...
    public class A_non_empty_queue ...
        public class that_is_not_full ...
            public void becomes_longer_when_value_enqueued() ...
            public void becomes_full_when_enqueued_up_to_capacity() ...
        public class that_is_full ...
            public void ignores_further_enqueued_values() ...
            public void becomes_non_full_when_dequeued() ...
        public void dequeues_values_in_order_enqueued() ...
```

```
namespace Queue_spec ...
    public class A_new_queue ...
        public void is_empty()
        public void preserves_positive_bounding_capacity() ...
        public void rejects_a_zero_bounding_capacity() ...
        public void rejects_a_negative_bounding_capacity() ...
    public class An_empty_queue ...
        public void dequeues_default_values()
        public void becomes_non_empty_when_value_enqueued() ...
    public class A_non_empty_queue ...
        public class that_is_not_full ...
            public void becomes_longer_when_value_enqueued() ...
            public void becomes_full_when_enqueued_up_to_capacity() ...
        public class that_is_full ...
            public void ignores_further_enqueued_values() ...
            public void becomes_non_full_when_dequeued() ...
        public void dequeues_values_in_order_enqueued() ...
```


LEVEL
DRIVE
OUTPUT
FUZZBOX
INPUT
9v dc
COOLTONE

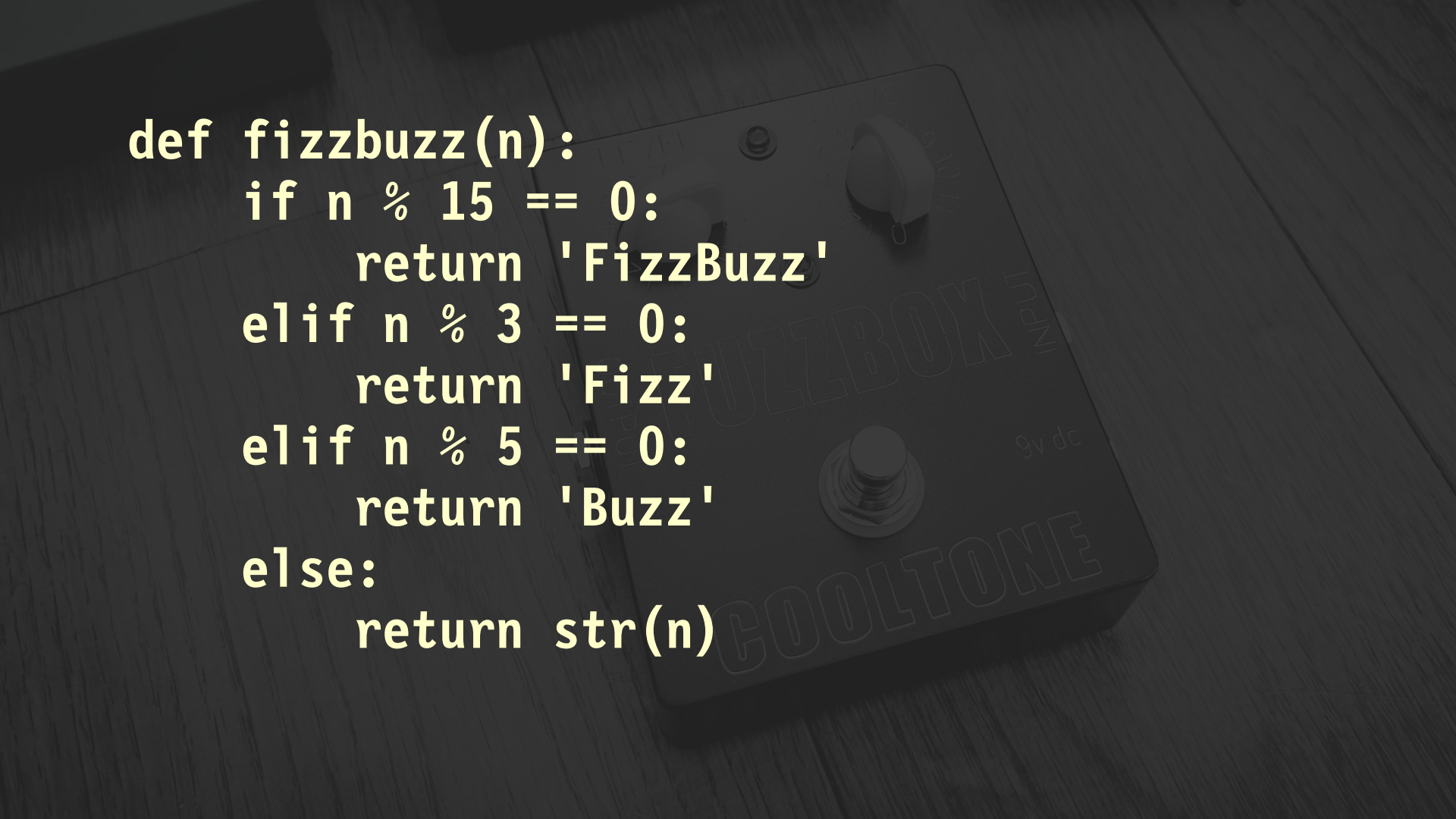
Richard Dalton
@richardadalton

FizzBuzz was invented to avoid the awkwardness of realising that nobody in the room can binary search an array.

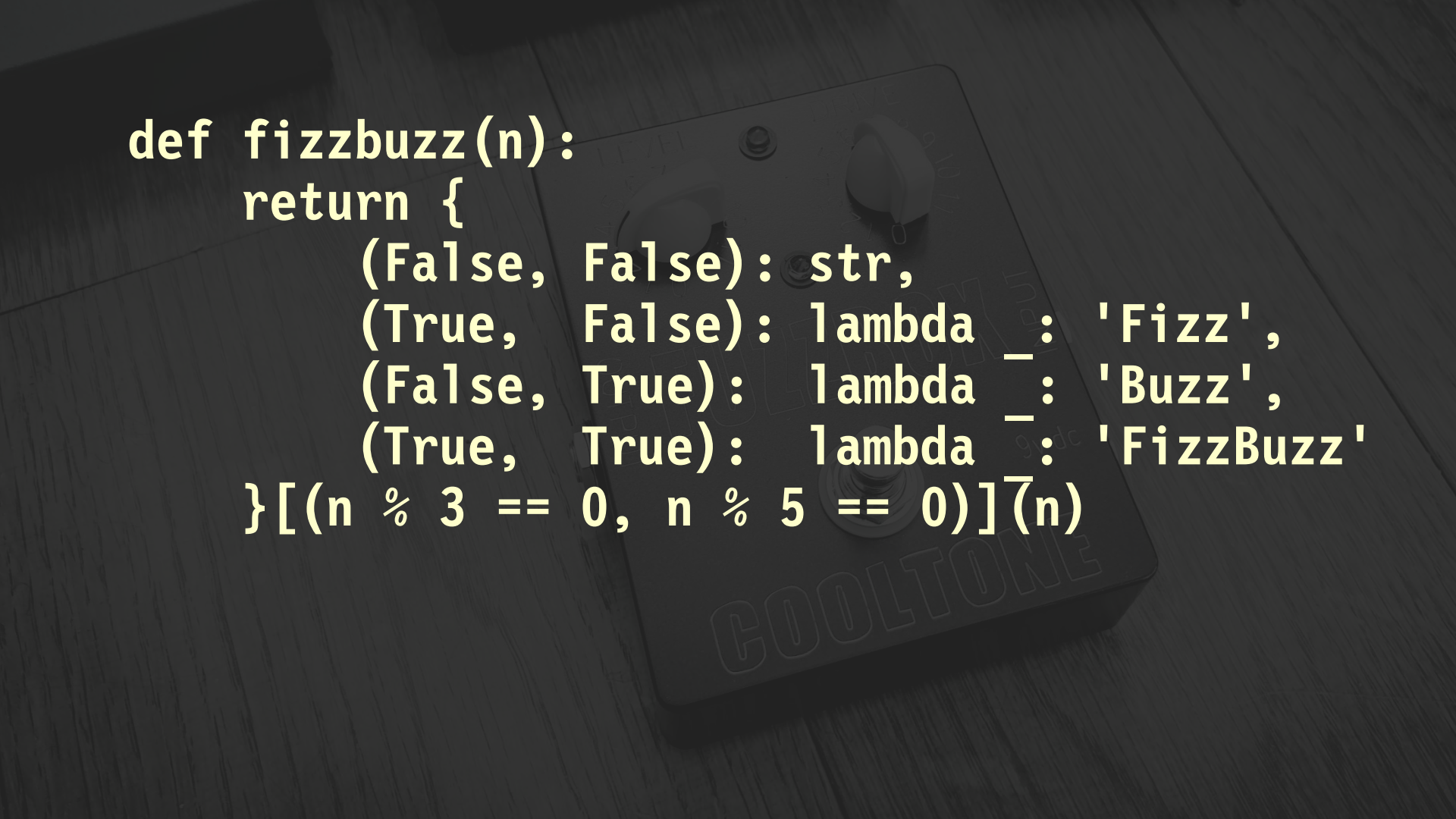
10:29 AM · Apr 24, 2015

13 15 Share this Tweet

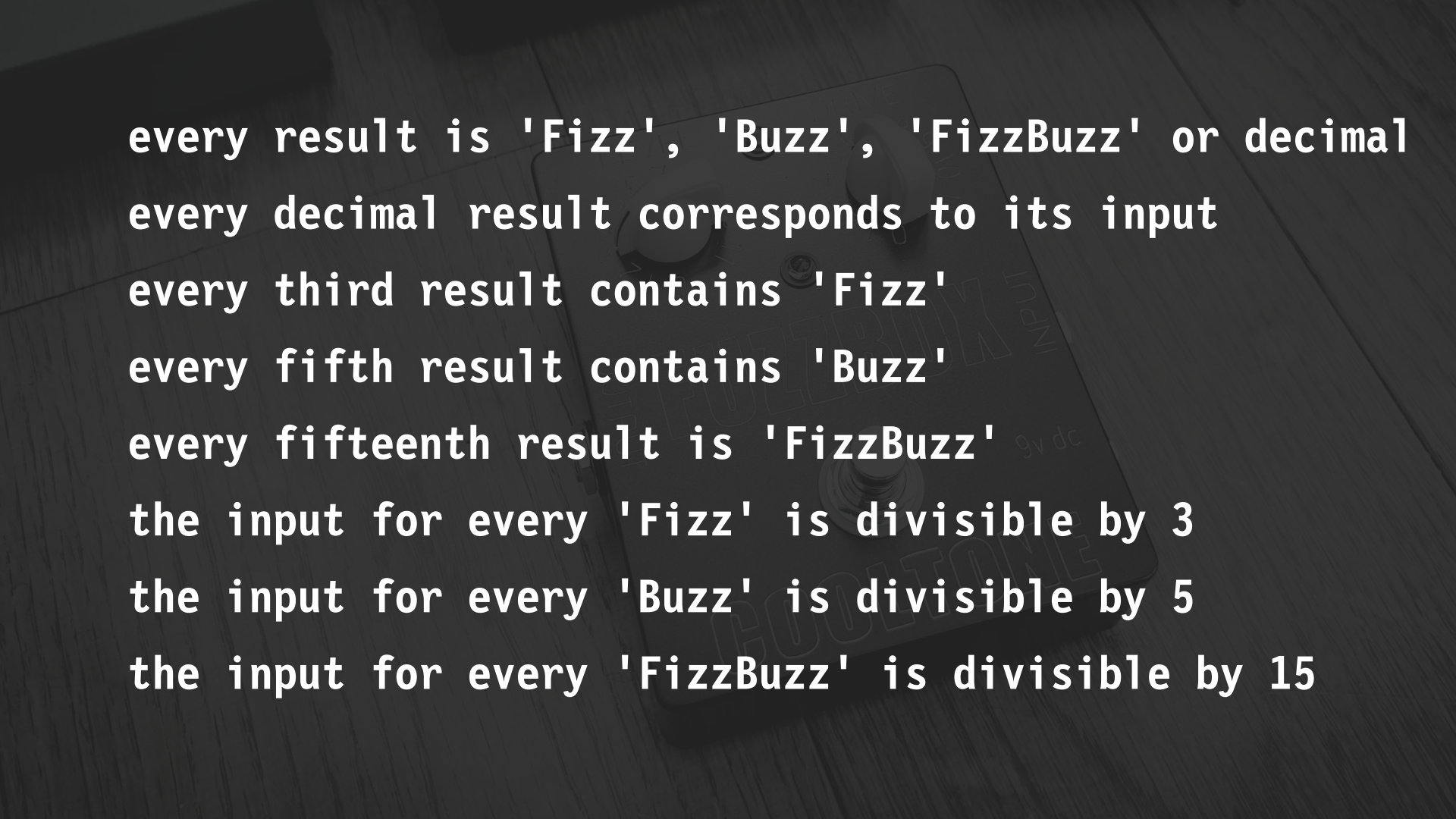
twitter.com/richardadalton/status/591534529086693376




```
def fizzbuzz(n):
    if n % 15 == 0:
        return 'FizzBuzz'
    elif n % 3 == 0:
        return 'Fizz'
    elif n % 5 == 0:
        return 'Buzz'
    else:
        return str(n)
```


def fizzbuzz(n):
return {
(False, False): str,
(True, False): lambda _: 'Fizz',
(False, True): lambda _: 'Buzz',
(True, True): lambda _: 'FizzBuzz'
}[(n % 3 == 0, n % 5 == 0)](n)

```
def test_that_result_for_1_is_1(n):
    assert fizzbuzz(1) == '1'

def test_that_result_for_2_is_2(n):
    assert fizzbuzz(2) == '2'

def test_that_result_for_3_is_Fizz(n):
    assert fizzbuzz(3) == 'Fizz'

def test_that_result_for_4_is_4(n):
    assert fizzbuzz(4) == '4'

def test_that_result_for_5_is_Buzz(n):
    assert fizzbuzz(5) == 'Buzz'

...
```



every result is 'Fizz', 'Buzz', 'FizzBuzz' or decimal
every decimal result corresponds to its input
every third result contains 'Fizz'
every fifth result contains 'Buzz'
every fifteenth result is 'FizzBuzz'
the input for every 'Fizz' is divisible by 3
the input for every 'Buzz' is divisible by 5
the input for every 'FizzBuzz' is divisible by 15

```
@mark.parametrize('n', range(1, 101))
def test_that_every_result_is_Fizz_Buzz_FizzBuzz_or_decimal(n):
    result = fizzbuzz(n)
    assert result in {'Fizz', 'Buzz', 'FizzBuzz'} or result.isdecimal()


@mark.parametrize('n', range(1, 101))
def test_that_every_decimal_result_corresponds_to_its_input(n):
    result = fizzbuzz(n)
    if result.isdecimal():
        assert int(result) == n
```

```
@mark.parametrize(
    'n, result',
    ((i, fizzbuzz(i)) for i in range(1, 101)))
def test_that_every_result_is_Fizz_Buzz_FizzBuzz_or_decimal(n, result):
    assert result in {'Fizz', 'Buzz', 'FizzBuzz'} or result.isdecimal()

@mark.parametrize(
    'n, result',
    ((i, fizzbuzz(i)) for i in range(1, 101)
     if fizzbuzz(i).isdecimal()))
def test_that_every_decimal_result_corresponds_to_its_input(n, result):
    assert int(result) == n
```

```
@mark.parametrize('n', range(3, 101, 3))
def test_that_every_third_result_contains_Fizz(n):
    assert 'Fizz' in fizzbuzz(n)

@mark.parametrize('n', range(5, 101, 5))
def test_that_every_fifth_result_contains_Buzz(n):
    assert 'Buzz' in fizzbuzz(n)

@mark.parametrize('n', range(15, 101, 15))
def test_that_every_fifteenth_result_is_FizzBuzz(n):
    assert fizzbuzz(n) == 'FizzBuzz'
```

```
@mark.parametrize(
    'n', (i for i in range(1, 101) if fizzbuzz(i) == 'Fizz'))
def test_that_the_input_for_every_Fizz_is_divisible_by_3(n):
    assert n % 3 == 0

@mark.parametrize(
    'n', (i for i in range(1, 101) if fizzbuzz(i) == 'Buzz'))
def test_that_the_input_for_every_Buzz_is_divisible_by_5(n):
    assert n % 5 == 0

@mark.parametrize(
    'n', (i for i in range(1, 101) if fizzbuzz(i) == 'FizzBuzz'))
def test_that_the_input_for_every_FizzBuzz_is_divisible_by_15(n):
    assert n % 15 == 0
```

```
def test_that_every_result_is_Fizz_Buzz_FizzBuzz_or_decimal(...):
    ...
def test_that_every_decimal_result_corresponds_to_its_input(...):
    ...
def test_that_every_third_result_contains_Fizz(...):
    ...
def test_that_every_fifth_result_contains_Buzz(...):
    ...
def test_that_every_fifteenth_result_is_FizzBuzz(...):
    ...
def test_that_the_input_for_every_Fizz_is_divisible_by_3(...):
    ...
def test_that_the_input_for_every_Buzz_is_divisible_by_5(...):
    ...
def test_that_the_input_for_every_FizzBuzz_is_divisible_by_15(...):
    ...
```

```
def test_that_every_result_is_Fizz_Buzz_FizzBuzz_or_decimal(...):
    ...
def test_that_every_decimal_result_corresponds_to_its_input(...):
    ...
def test_that_every_third_result_contains_Fizz(...):
    ...
def test_that_every_fifth_result_contains_Buzz(...):
    ...
def test_that_every_fifteenth_result_is_FizzBuzz(...):
    ...
def test_that_the_input_for_every_Fizz_is_divisible_by_3(...):
    ...
def test_that_the_input_for_every_Buzz_is_divisible_by_5(...):
    ...
def test_that_the_input_for_every_FizzBuzz_is_divisible_by_15(...):
    ...
```

Algorithms + Data Structures = Programs

Niklaus Wirth

Structure + Interpretation = Programs