Lecture 10 - June 5

TDD with JUnit

JUnit Test Method vs. Method Under Test

TDD & Regression Testing

JUnit Test: An Exception Expected or Not

Announcements/Reminders

- Today's class: notes template posted
- ProgTest1 next Friday JUN 13
- ProgTest1 guide (policies & requirements) released
  - + PracticeTest1 & Survey on Review Session Time
  - + ProgTest0 marks & feedback (or MON, JUN 9 latest)
- Priorities:
  - + Lab1 solution, Lab2
  - + Slides on Classes and Objects
  - + Slides on Exceptions

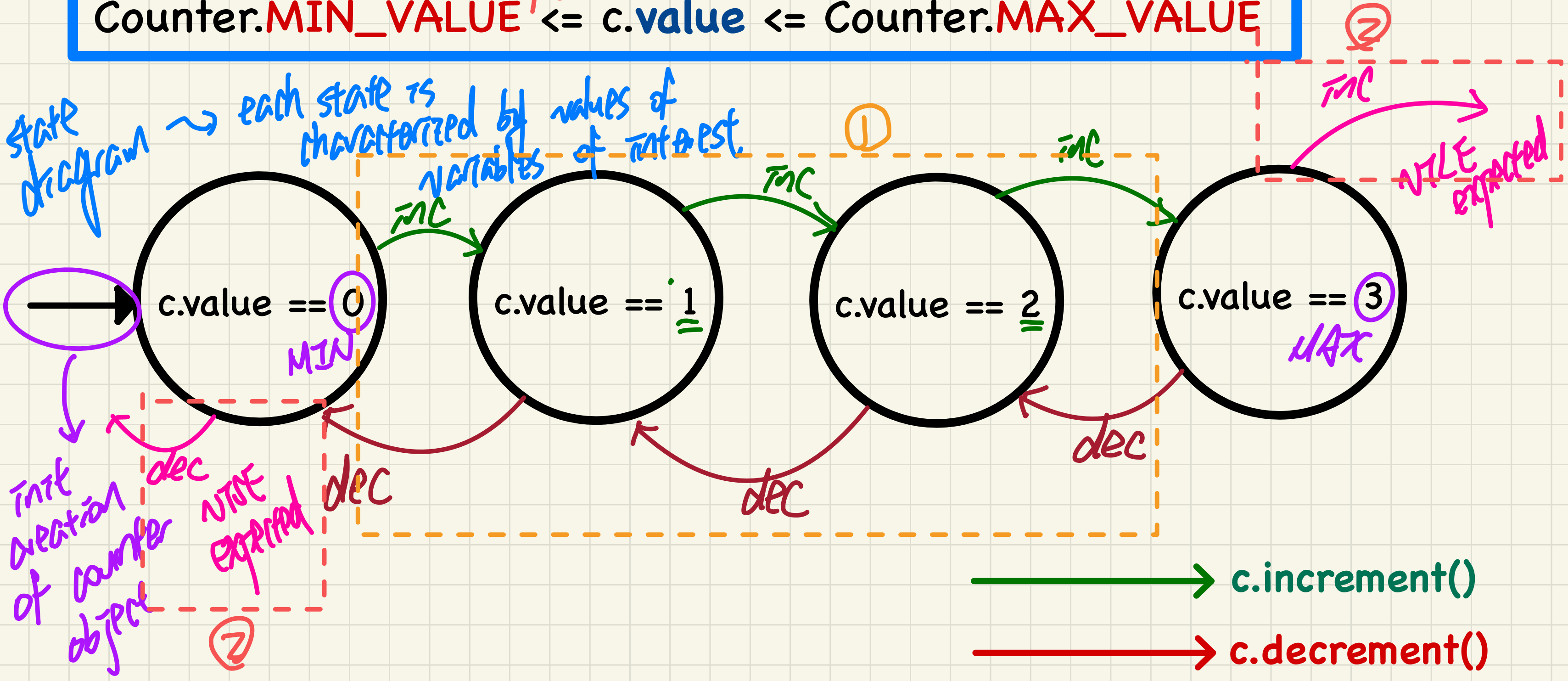
① normal scenarios to test ② abnormal scenarios to test

Coming Up with Test Cases: A Single, Bounded Variable

② ~~OR~~ exception occurred → fail ; exception not occurred → pass

Boundries: ② exception expected occurred → pass not occurred → fail

Counter.MIN_VALUE <= c.value <= Counter.MAX_VALUE

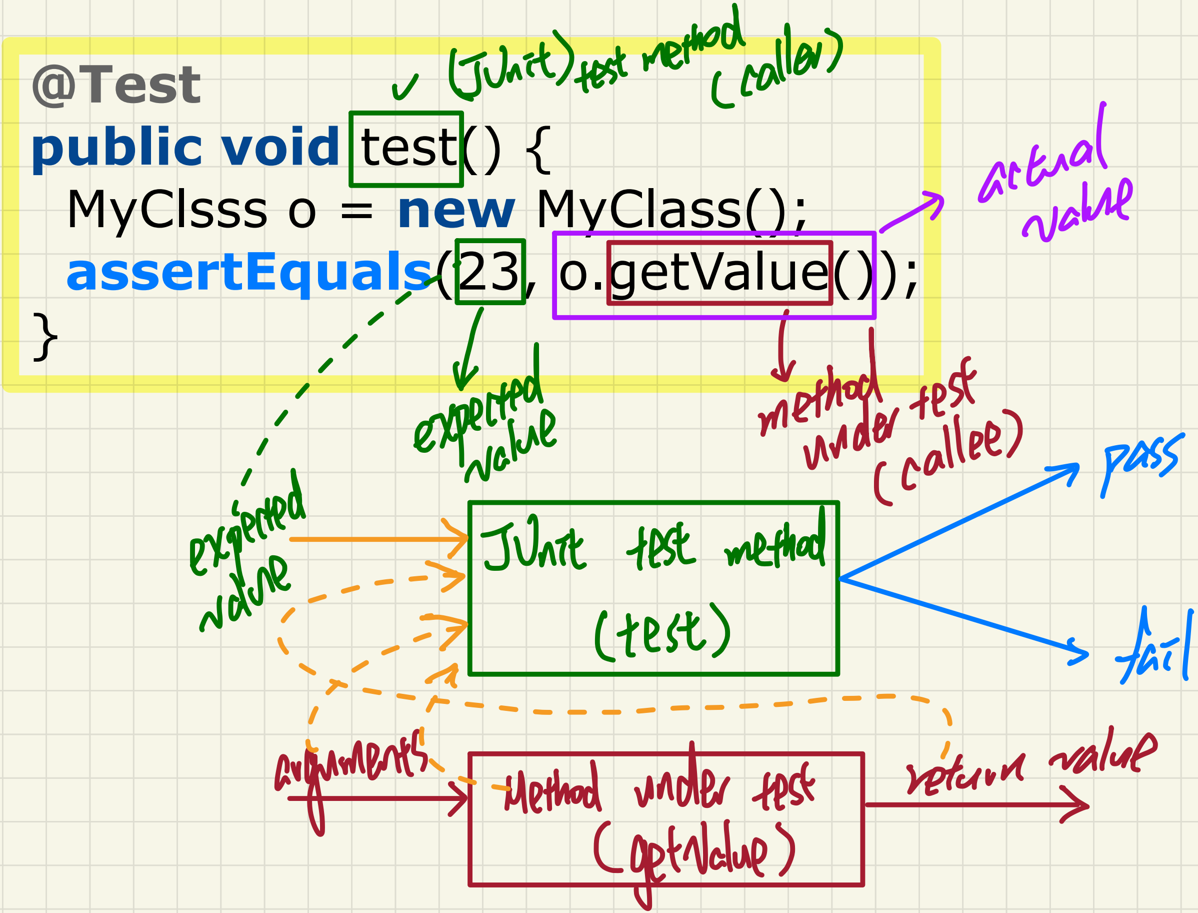


state diagram ~> each state is characterized by values of variables of interest
init creation of counter object
c.value == 0
MIN
inc
c.value == 1
inc
c.value == 2
inc
c.value == 3
MAX
dec
dec
dec
dec
VTSE expected
inc
VTLE expected
①
②
②

c.increment()

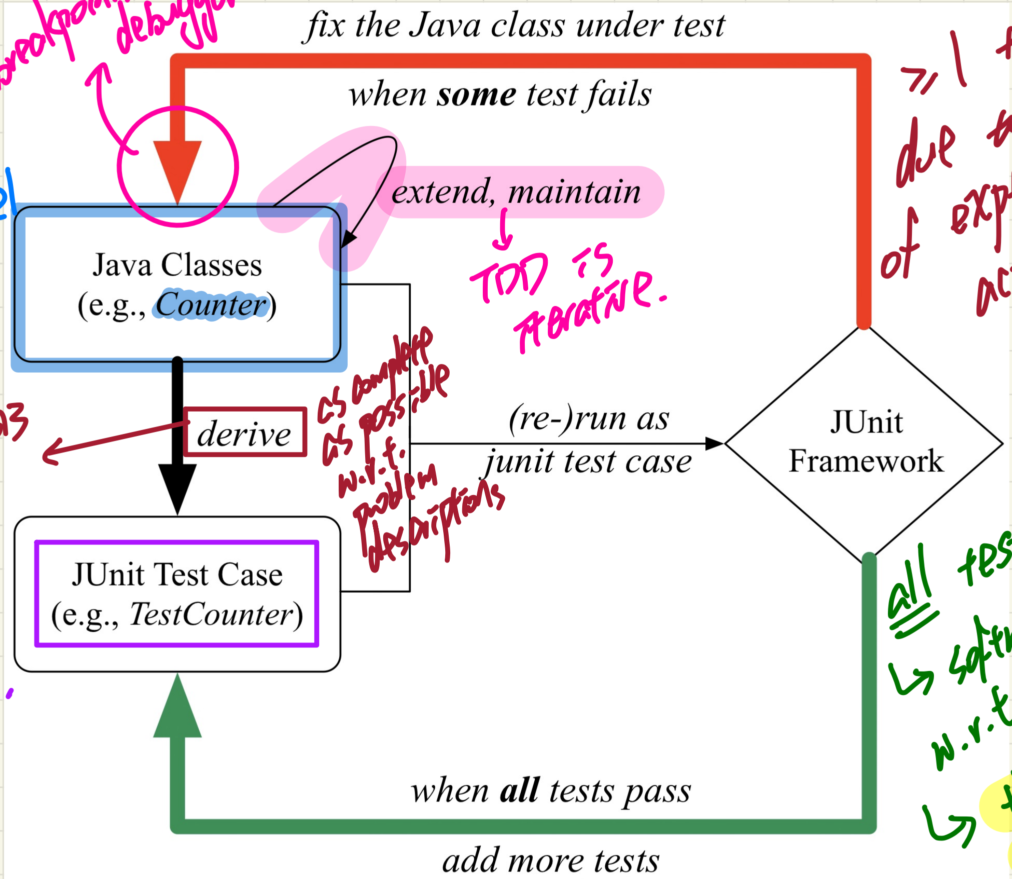
c.decrement()

JUnit Test Method vs. Method Under Test



```
@Test
public void test() {
  MyClsss o = new MyClass();
  assertEquals(23, o.getValue());
}
```

Test-Driven Development (TDD): Regression Testing



fix the Java class under test

when **some** test fails

extend, maintain

Java Classes
(e.g., *Counter*)

derive

JUnit Test Case
(e.g., *TestCounter*)

(re-)run as
junit test case

JUnit
Framework

when **all** tests pass

add more tests

breakpoints & debugger.

model

incremental development

⇒ re-run all tests after each small, logical unit is implemented.

EECS4313

as complete as possible w.r.t. problem descriptions

↓ TDD is iterative.

≥ 1 test failed due to a mismatch of expected and actual values.

all tests pass

↳ software is "correct" w.r.t. the tests run

↳ the more aspects being tested, the better.

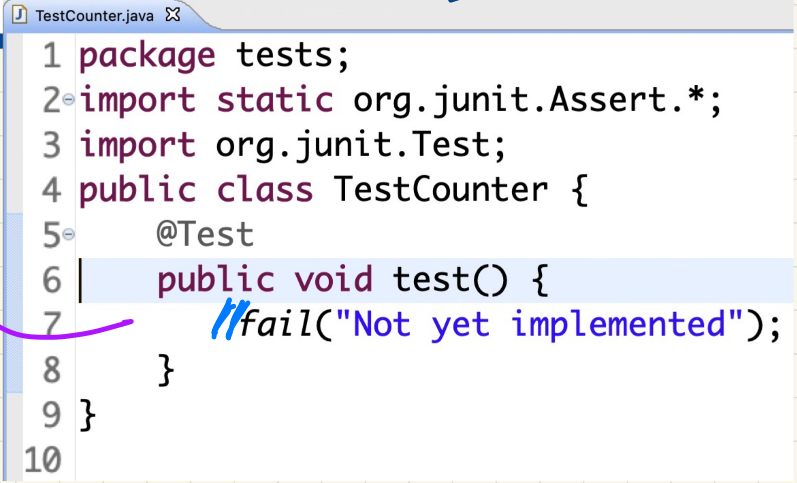
A Default Test Case that Fails

slide 16

The result of running a test is considered:

- ***Failure*** if either
  - an ***assertion failure*** (e.g., caused by **`fail`**, **`assertTrue`**, **`assertEquals`**) occurs
  - an unexpected ***exception*** (e.g., *`NullPointerException`*, *`ArrayIndexOutOfBoundException`*) thrown
- ***Success*** if neither ***assertion failures*** nor (unexpected) ***exceptions*** occur.

slide 19



```java
package tests;
import static org.junit.Assert.*;
import org.junit.Test;
public class TestCounter {
    @Test
    public void test() {
        //fail("Not yet implemented");
    }
}
```

leaving the body of test method empty will pass the test trivially

✓

Q: What is the easiest way to making this test **pass**?

Examples: JUnit Assertions (1)

slid 23

Consider the following class:

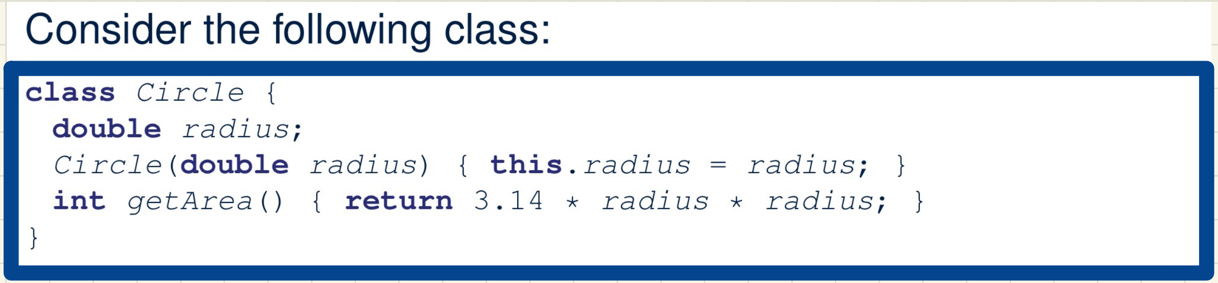
```
public class Point {
  private int x; private int y;
  public Point(int x, int y) { this.x = x; this.y = y; }
  public int getX() { return this.x; }
  public int getY() { return this.y; }
}
```

Then consider these assertions. Do they ***pass*** or ***fail***?

```
Point p;
assertNull(p);  [redacted]
assertTrue(p == null);  [redacted]
assertFalse(p != null);  [redacted]
assertEquals(3, p.getX());  [redacted] [redacted]
p = new Point(3, 4);
assertNull(p);  [redacted]
assertTrue(p == null);  [redacted]
assertFalse(p != null);  [redacted]
assertEquals(3, p.getX());  [redacted]
assertTrue(p.getX() == 3 && p.getY() == 4);  [redacted]
```

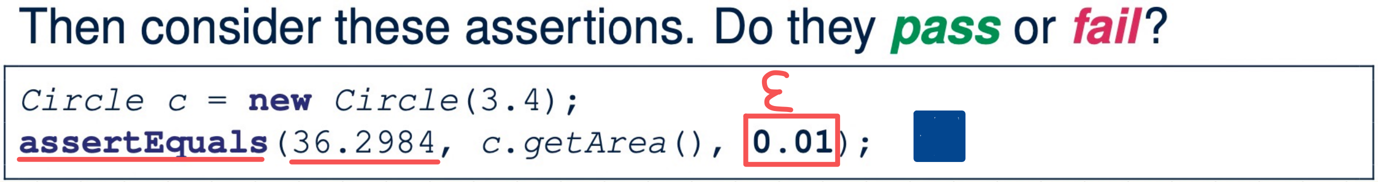
Examples: JUnit Assertions (2)

slide 24



Consider the following class:

```
class Circle {
  double radius;
  Circle(double radius) { this.radius = radius; }
  int getArea() { return 3.14 * radius * radius; }
}
```



Then consider these assertions. Do they *pass* or *fail*?

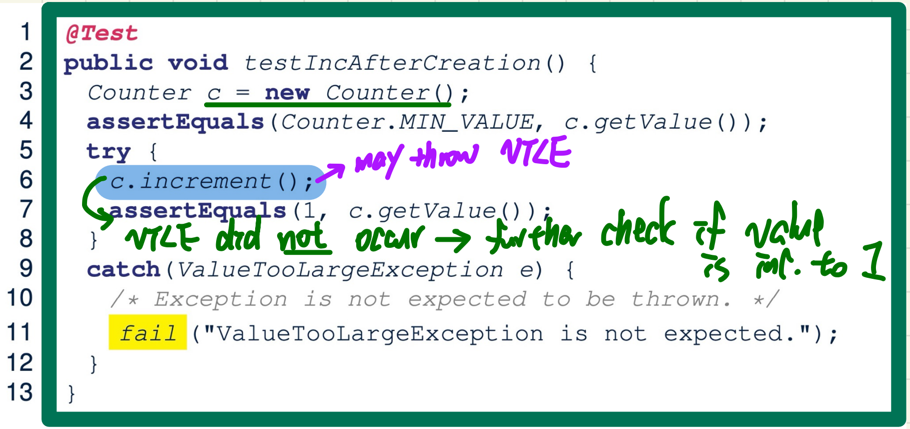
```
                                      ε
Circle c = new Circle(3.4);
assertEquals(36.2984, c.getArea(), 0.01);
```

[redacted]

Test

- → **Expectation**: certain exception occurs
  - ↳ that kind of exception occurs → pass
  - ↳ that kind of exception does not occur → fail
- → **Expectation**: certain exception does not occur
  - ↳ that kind of exception occurs → fail
  - ↳ that kind of exception does not occur → pass

JUnit: An Exception Not Expected



```
@Test
public void testIncAfterCreation() {
  Counter c = new Counter();
  assertEquals(Counter.MIN_VALUE, c.getValue());
  try {
    c.increment();
    assertEquals(1, c.getValue());
  }
  catch(ValueTooLargeException e) {
    /* Exception is not expected to be thrown. */
    fail ("ValueTooLargeException is not expected.");
  }
}
```

may throw VTLE

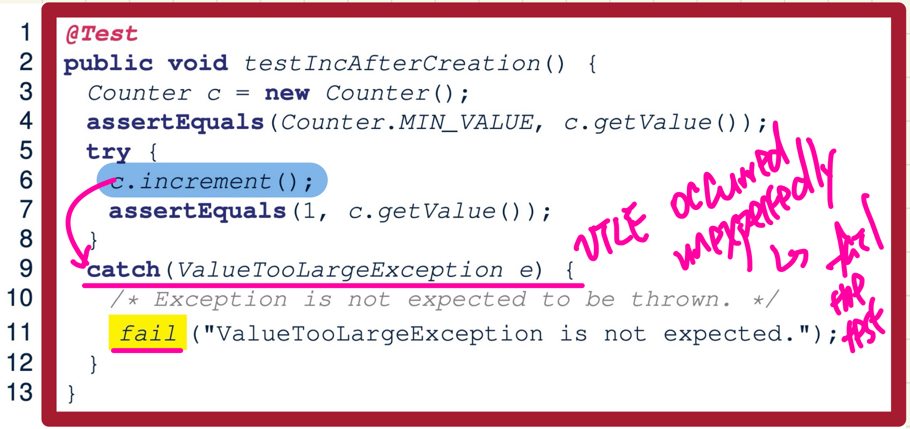
VTLE did not occur → further check if value is inc. to 1

What if increment is implemented correctly?

Expected Behaviour:

Calling c.increment()
when c.value is 0 should **not**
trigger a ValueTooLargeException

not expected to happen



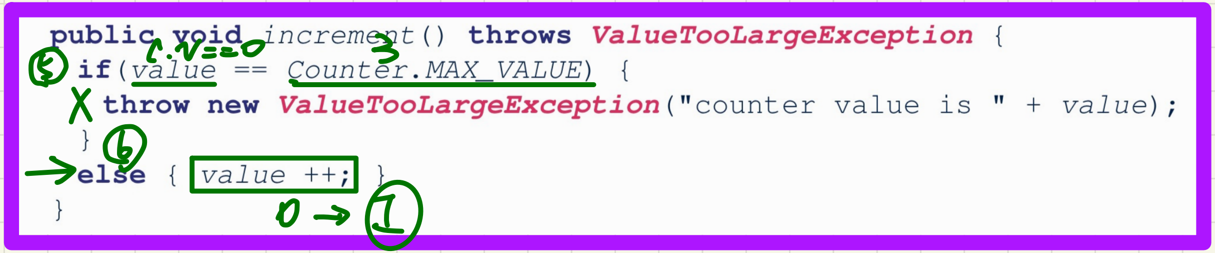
```
@Test
public void testIncAfterCreation() {
  Counter c = new Counter();
  assertEquals(Counter.MIN_VALUE, c.getValue());
  try {
    c.increment();
    assertEquals(1, c.getValue());
  }
  catch(ValueTooLargeException e) {
    /* Exception is not expected to be thrown. */
    fail ("ValueTooLargeException is not expected.");
  }
}
```

VTLE occurred unexpectedly ↳ fail the test

What if increment is implemented incorrectly?

e.g., It throws VTLE when
c.value < Counter.MAX_VALUE

Running JUnit Test 1 on Correct Implementation



```
public void increment() throws ValueTooLargeException {
  if(value == Counter.MAX_VALUE) {
    throw new ValueTooLargeException("counter value is " + value);
  }
  else { value ++; }
}
```

c.v==0

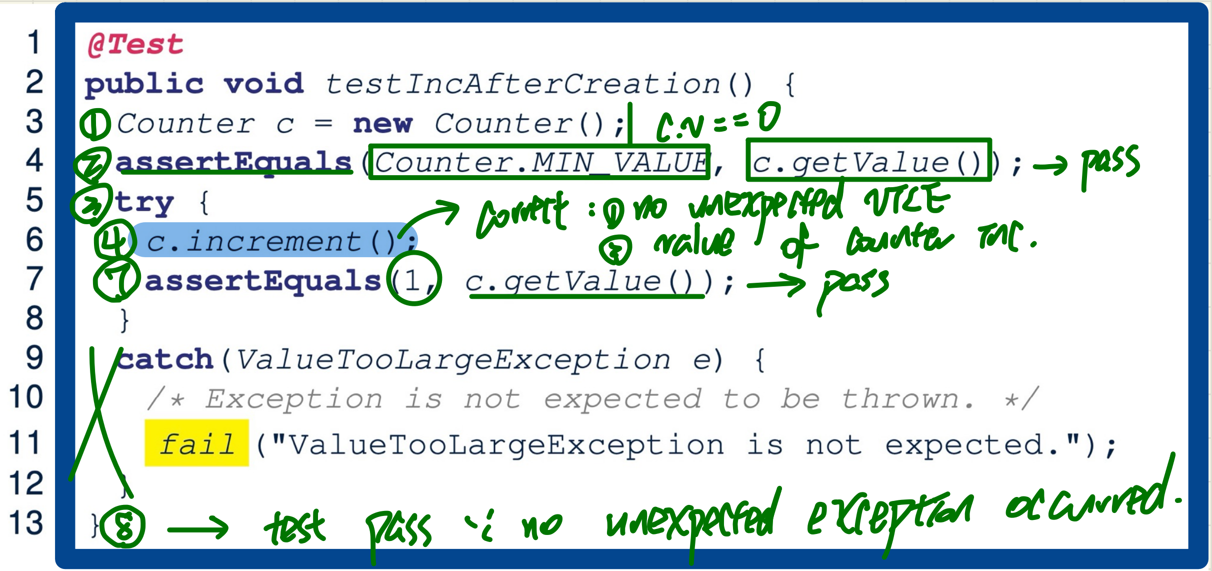
3

⑤

X

⑥

0 → ①



```
@Test
public void testIncAfterCreation() {
  Counter c = new Counter();
  assertEquals(Counter.MIN_VALUE, c.getValue());
  try {
    c.increment();
    assertEquals(1, c.getValue());
  }
  catch(ValueTooLargeException e) {
    /* Exception is not expected to be thrown. */
    fail ("ValueTooLargeException is not expected.");
  }
}
```

① c.v==0

② → pass

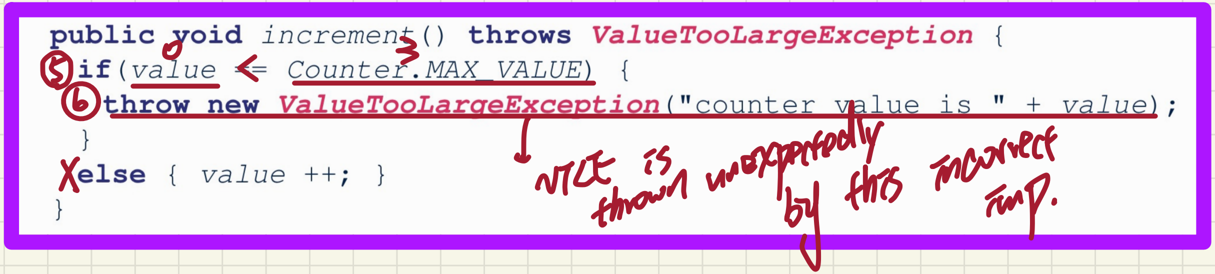
③

④ correct : ① no unexpected VTLE ② value of counter inc.

⑦ → pass

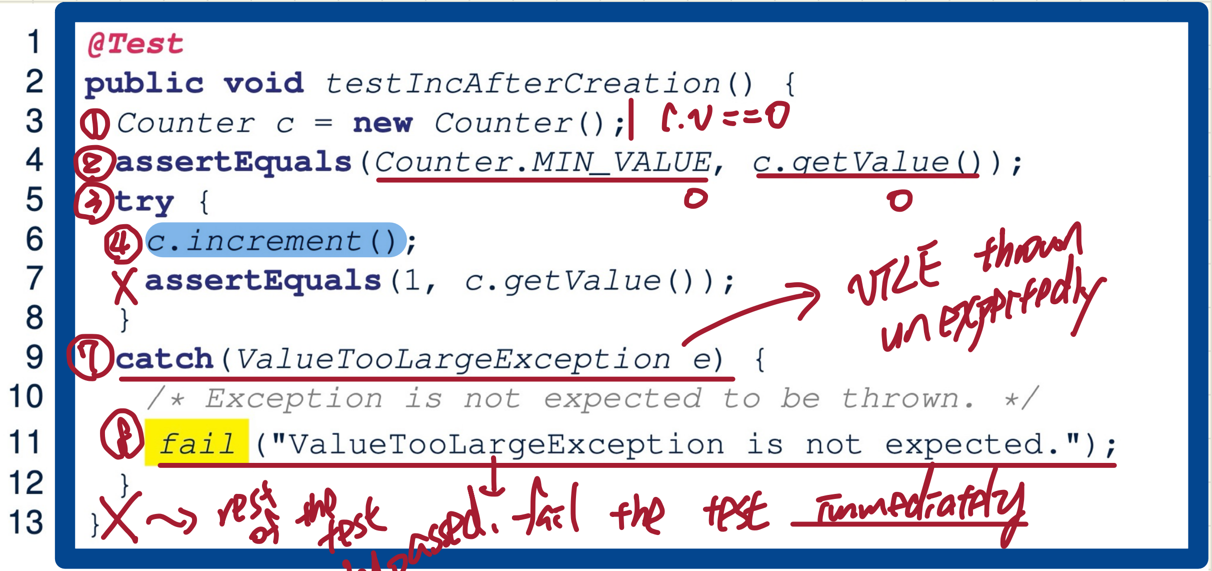
⑧ → test pass ∵ no unexpected exception occurred.

Running JUnit Test 1 on Incorrect Implementation



```
public void increment() throws ValueTooLargeException {
                 0        3
⑤if(value <= Counter.MAX_VALUE) {
 ⑥throw new ValueTooLargeException("counter value is " + value);
  }
 Xelse { value ++; }
}
```

VTLE is thrown unexpectedly by this incorrect imp.

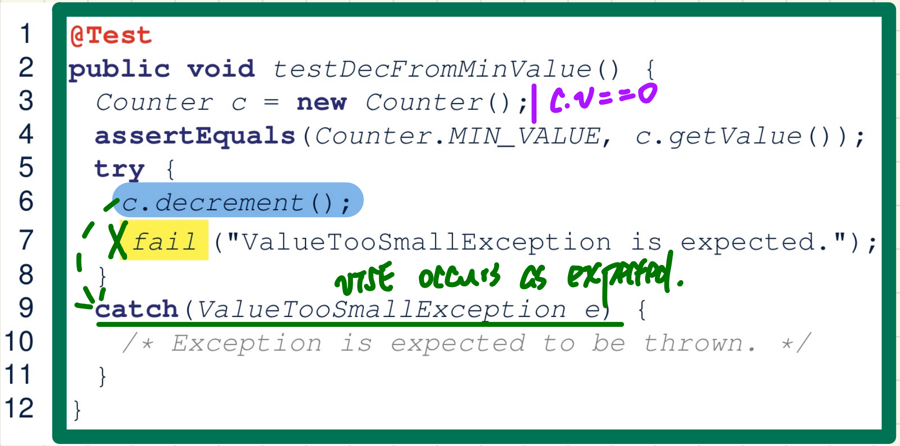


```
@Test
public void testIncAfterCreation() {
①Counter c = new Counter();| c.v == 0
②assertEquals(Counter.MIN_VALUE, c.getValue());
                              0            0
③try {
  ④c.increment();
  XassertEquals(1, c.getValue());
 }
⑦catch(ValueTooLargeException e) {
  /* Exception is not expected to be thrown. */
  ⑧fail ("ValueTooLargeException is not expected.");
 }
}X ~> rest of the test bypassed.
```

VTLE thrown unexpectedly

fail the test immediately

JUnit: An Exception Expected



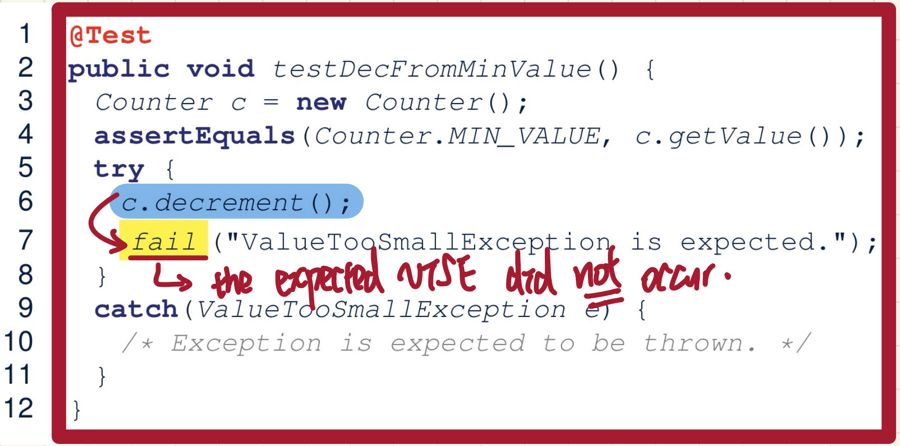
```java
@Test
public void testDecFromMinValue() {
  Counter c = new Counter(); | c.v==0
  assertEquals(Counter.MIN_VALUE, c.getValue());
  try {
    c.decrement();
    fail("ValueTooSmallException is expected.");
  }
  catch(ValueTooSmallException e) {
    /* Exception is expected to be thrown. */
  }
}
```

VTSE occurs as expected.

What if decrement is implemented correctly?

Expected Behaviour:

Calling c.decrement()
when c.value is 0 should
trigger a ValueTooSmallException.



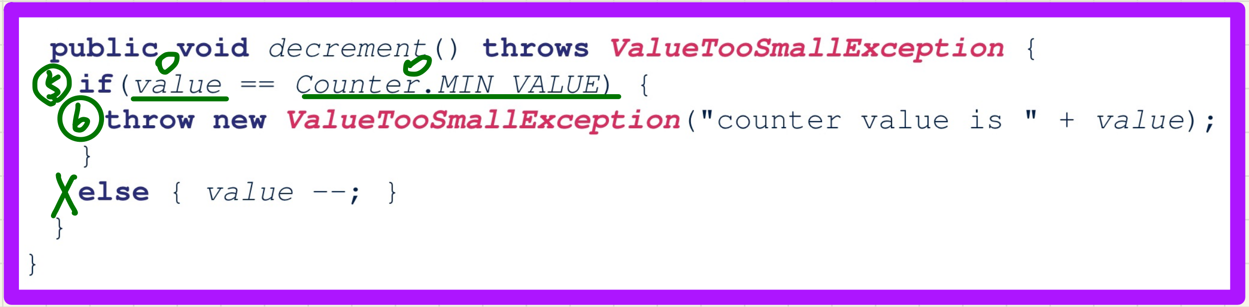
```java
@Test
public void testDecFromMinValue() {
  Counter c = new Counter();
  assertEquals(Counter.MIN_VALUE, c.getValue());
  try {
    c.decrement();
    fail("ValueTooSmallException is expected.");
  }
  catch(ValueTooSmallException e) {
    /* Exception is expected to be thrown. */
  }
}
```

↳ the expected VTSE did not occur.

What if decrement is implemented incorrectly?

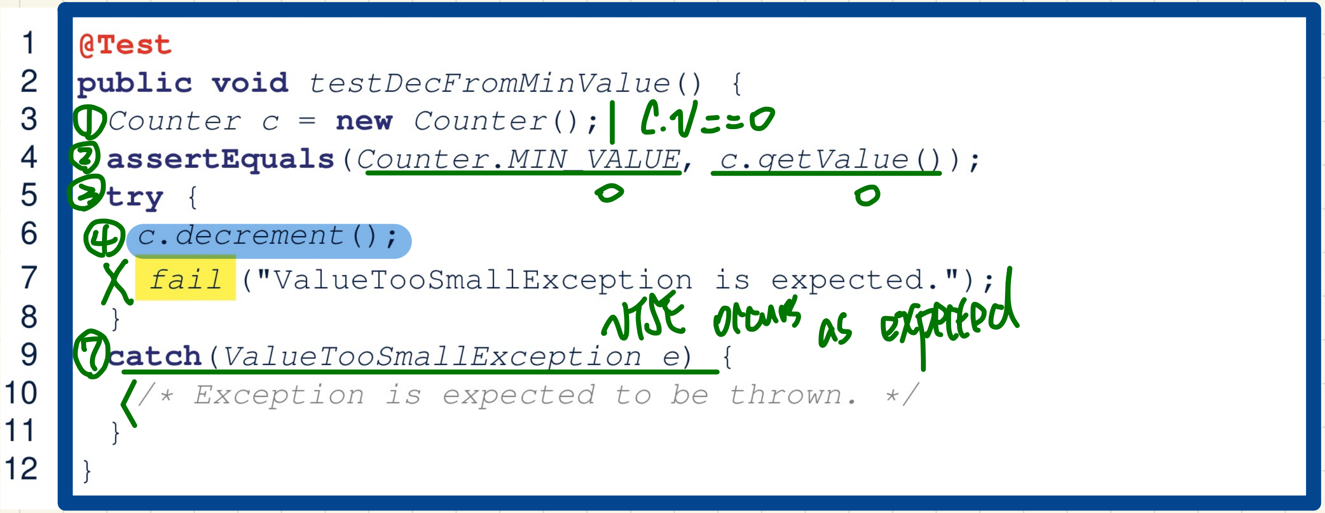
e.g., It only throws VTSE when
c.value < Counter.MIN_VALUE

Running JUnit Test 2 on Correct Implementation



```
 public void decrement() throws ValueTooSmallException {
⑤ if(value == Counter.MIN_VALUE) {
  ⑥throw new ValueTooSmallException("counter value is " + value);
   }
 X else { value --; }
  }
}
```

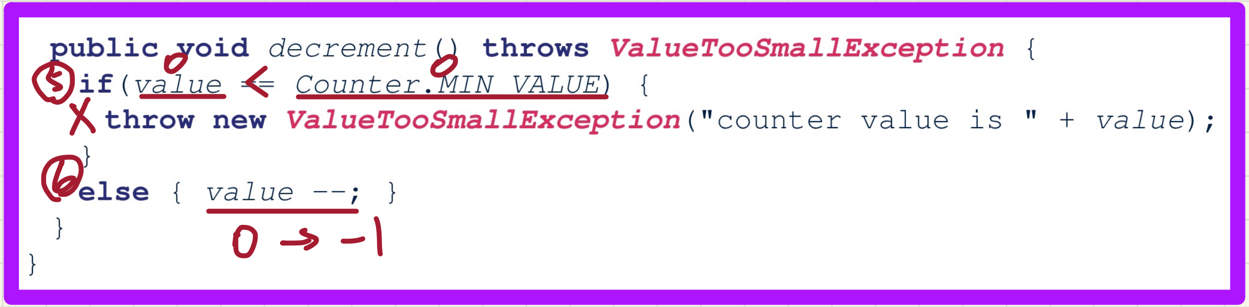
0 0



```
@Test
public void testDecFromMinValue() {
①Counter c = new Counter(); | c.v==0
②assertEquals(Counter.MIN_VALUE, c.getValue());
                       0                 0
③try {
 ④ c.decrement();
  X fail ("ValueTooSmallException is expected.");
  }
⑦catch(ValueTooSmallException e) {
  ✓/* Exception is expected to be thrown. */
  }
}
```

VTSE occurs as expected

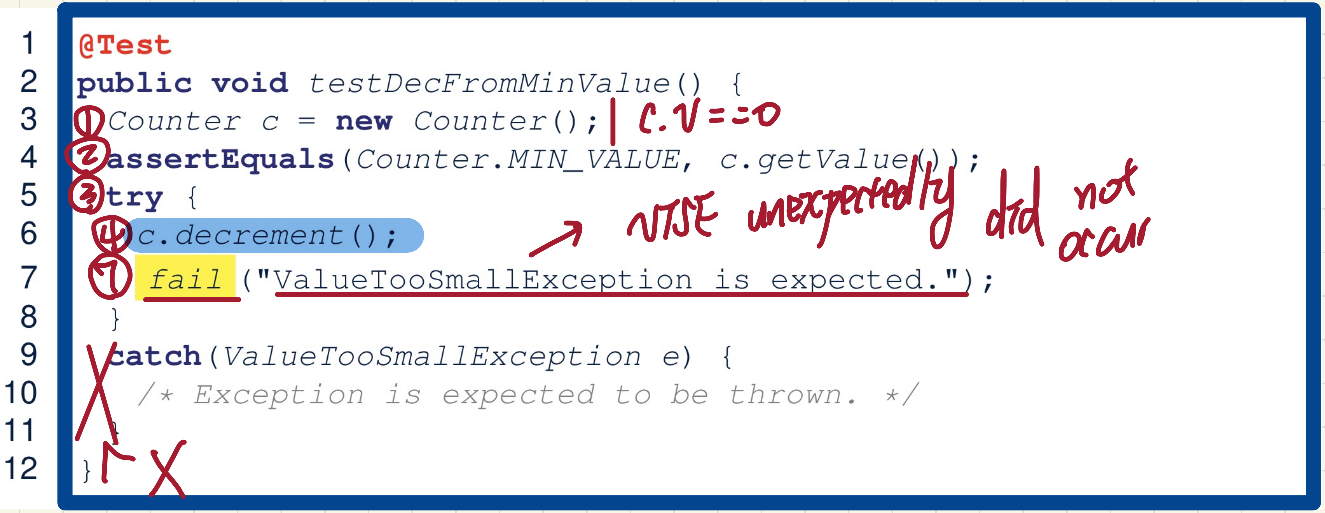
Running JUnit Test 2 on Incorrect Implementation



```java
public void decrement() throws ValueTooSmallException {
  if(value <= Counter.MIN_VALUE) {
    throw new ValueTooSmallException("counter value is " + value);
  }
  else { value --; }
 }
}
```

⑤ ⑥ 0 0 <

0 → -1



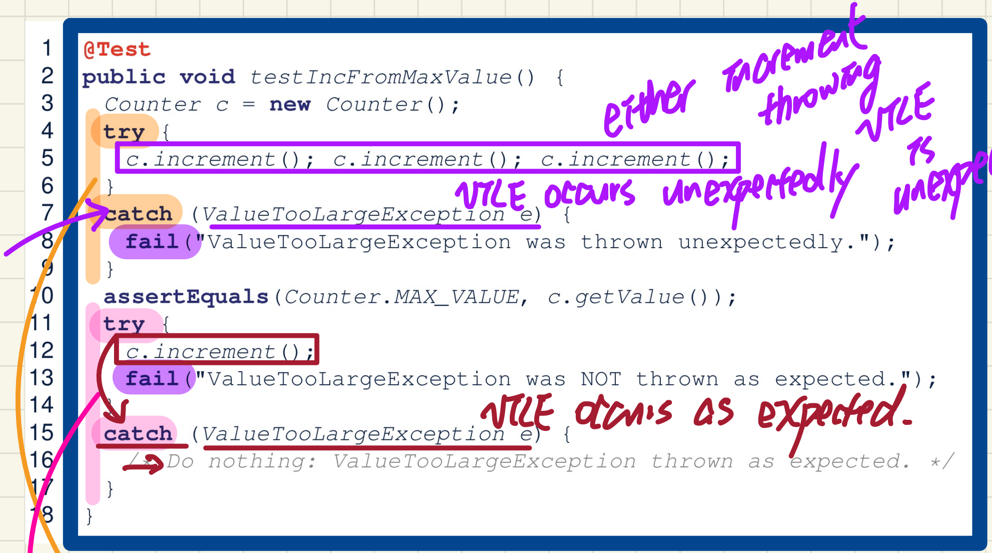
```java
@Test
public void testDecFromMinValue() {
  Counter c = new Counter();
  assertEquals(Counter.MIN_VALUE, c.getValue());
  try {
    c.decrement();
    fail ("ValueTooSmallException is expected.");
  }
  catch(ValueTooSmallException e) {
    /* Exception is expected to be thrown. */
  }
}
```

① ② ③ ④ ⑦

c.v == 0

VTSE unexpectedly did not occur

JUnit: Exception Sometimes Expected, Somtimes Not



```
@Test
public void testIncFromMaxValue() {
  Counter c = new Counter();
  try {
    c.increment(); c.increment(); c.increment();
  }
  catch (ValueTooLargeException e) {
    fail("ValueTooLargeException was thrown unexpectedly.");
  }
  assertEquals(Counter.MAX_VALUE, c.getValue());
  try {
    c.increment();
    fail("ValueTooLargeException was NOT thrown as expected.");
  }
  catch (ValueTooLargeException e) {
    /* Do nothing: ValueTooLargeException thrown as expected. */
  }
}
```

either increment throwing VTLE is unexpected

VTLE occurs unexpectedly

VTLE occurs as expected.

VTLE not expected

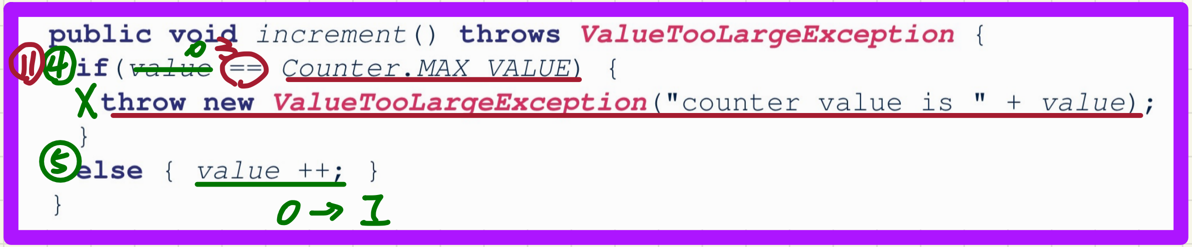
VTLE expected

Expected Behaviour:

Calling c.increment() 3 times to reach c's max **should not** trigger any ValueTooLargeException.

Calling c.increment() when c is already at its max **should** trigger a ValueTooLargeException

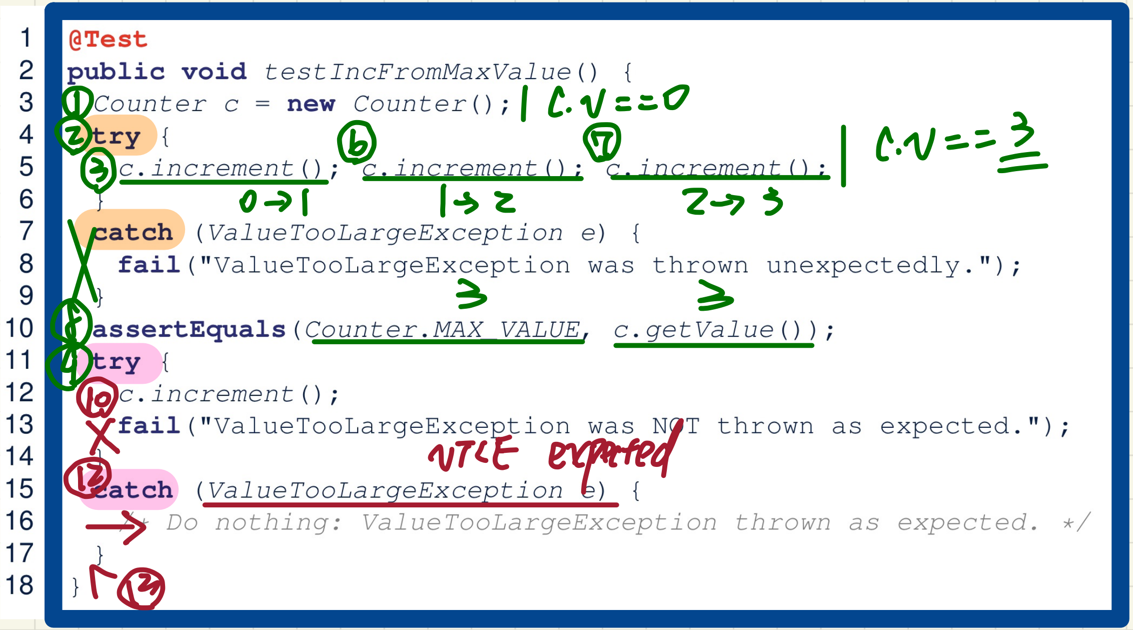
Running JUnit Test 3 on Correct Implementation



```
public void increment() throws ValueTooLargeException {
  if(value == Counter.MAX_VALUE) {
    throw new ValueTooLargeException("counter value is " + value);
  }
  else { value ++; }
}
```

0 3

0 → 1



```
@Test
public void testIncFromMaxValue() {
  Counter c = new Counter();
  try {
    c.increment(); c.increment(); c.increment();
  }
  catch (ValueTooLargeException e) {
    fail("ValueTooLargeException was thrown unexpectedly.");
  }
  assertEquals(Counter.MAX_VALUE, c.getValue());
  try {
    c.increment();
    fail("ValueTooLargeException was NOT thrown as expected.");
  }
  catch (ValueTooLargeException e) {
    /* Do nothing: ValueTooLargeException thrown as expected. */
  }
}
```

| c.v == 0

| c.v == 3

0 → 1   1 → 2   2 → 3

3   3

VTLE expected

PASS