Lecture 10 - Oct. 8

TDD with JUnit, Object Equality

JUnit Test: Exception Expected vs. Not

Using Loops in JUnit Test Methods

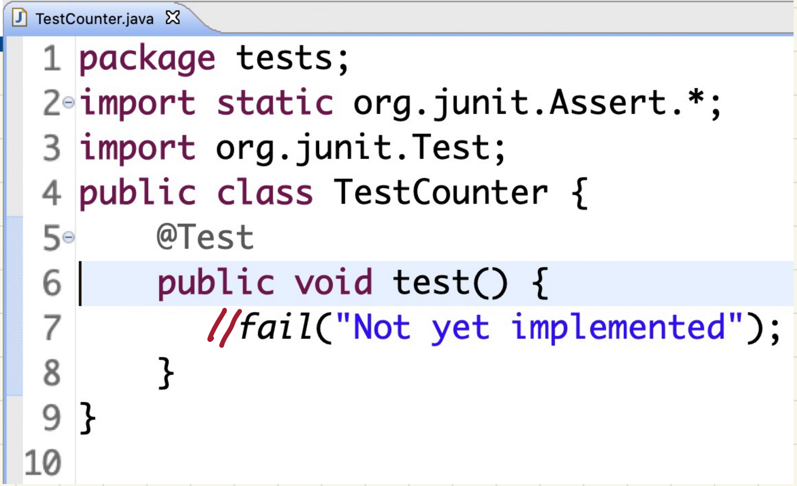
Announcements/Reminders

- ProgTest1 tomorrow
- ProgTest1 review session materials released
- Written Test 1 results released
- Lab1 solution released
- Lab2 released

A Default Test Case that Fails

The result of running a test is considered:

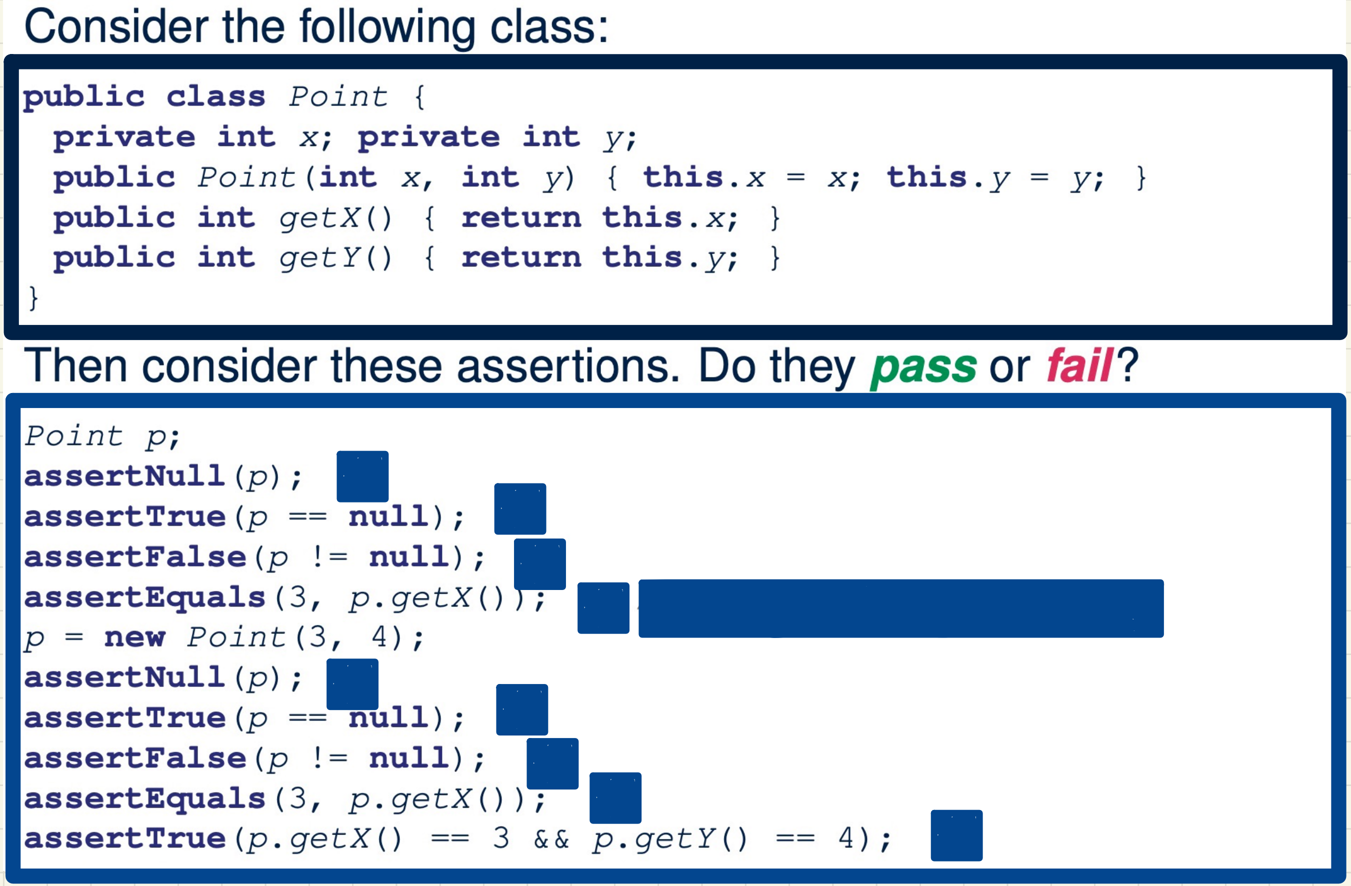
- *Failure* if either
  - an ***assertion failure*** (e.g., caused by **`fail`**, **`assertTrue`**, **`assertEquals`**) occurs
  - an unexpected ***exception*** (e.g., *`NullPointerException`*, *`ArrayIndexOutOfBoundException`*) thrown
- ***Success*** if neither ***assertion failures*** nor (unexpected)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



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

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

Examples: JUnit Assertions (1)



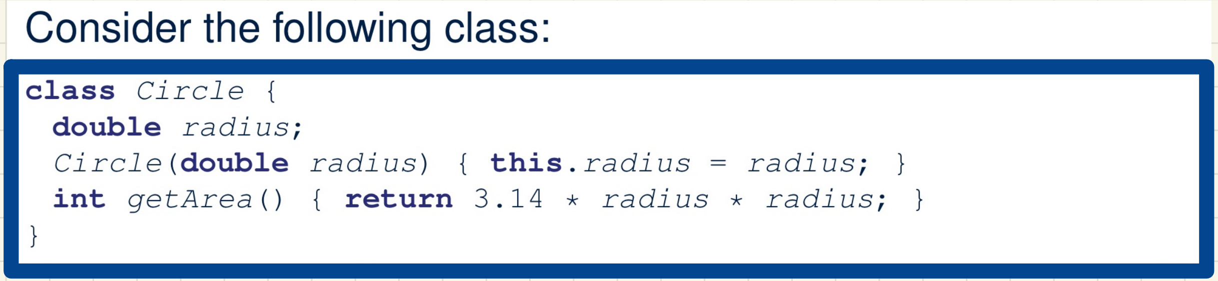
Consider the following class:

```
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*?

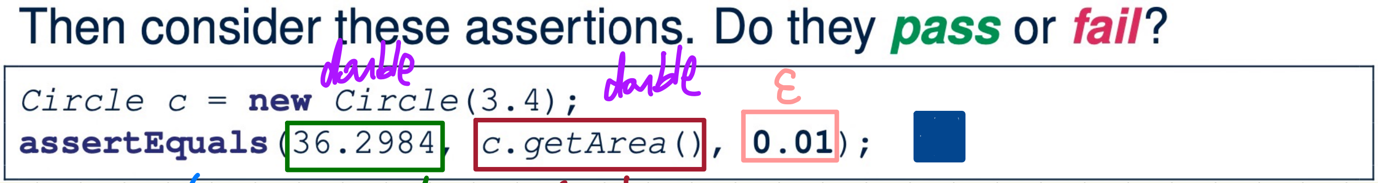
```
Point p;
assertNull(p); [redacted]
assertTrue(p == null); [redacted]
assertFalse(p != null); [redacted]
assertEquals(3, p.getX()); [redacted] [redacted]
p = new Point(3, 4);
assertNull(p); [redacted]
assertTrue(p == null); [redacted]
assertFalse(p != null); [redacted]
assertEquals(3, p.getX()); [redacted]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

Examples: JUnit Assertions (2)



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*?

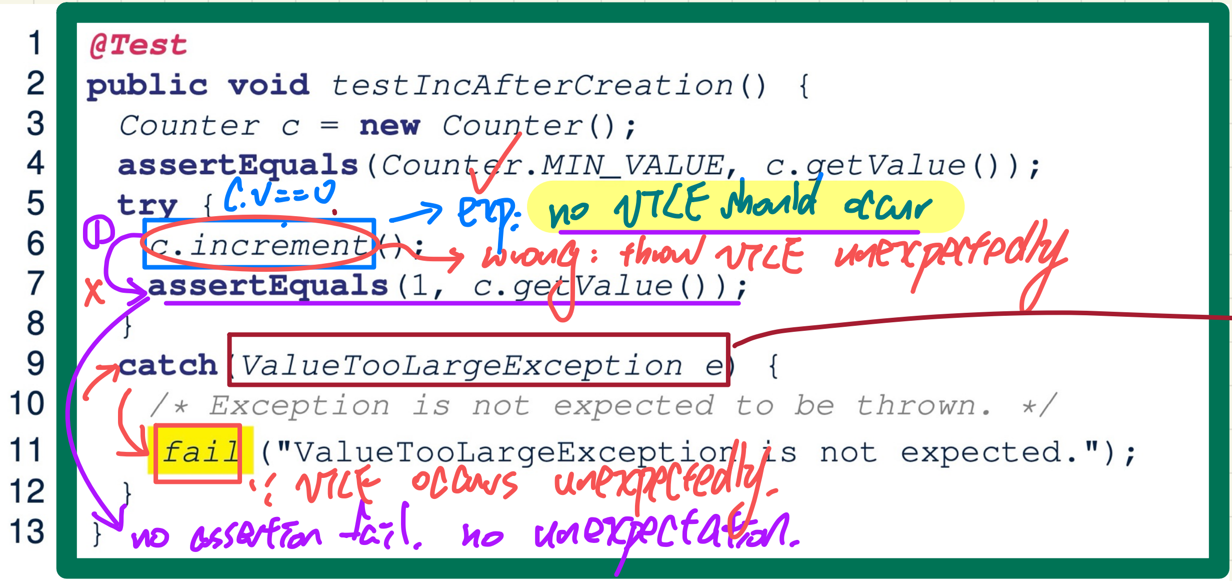
double double ε

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

expected actual

$$\text{expected} - \varepsilon \leq \text{actual} \leq \text{expected} + \varepsilon$$

public void increment() throws VTLE ...



```
@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.");
  }
}
```

c.v==0

exp: no VTLE should occur

wrong: throw VTLE unexpectedly

①

X

: VTLE occurs unexpectedly.

no assertion fail. no unexpectation.

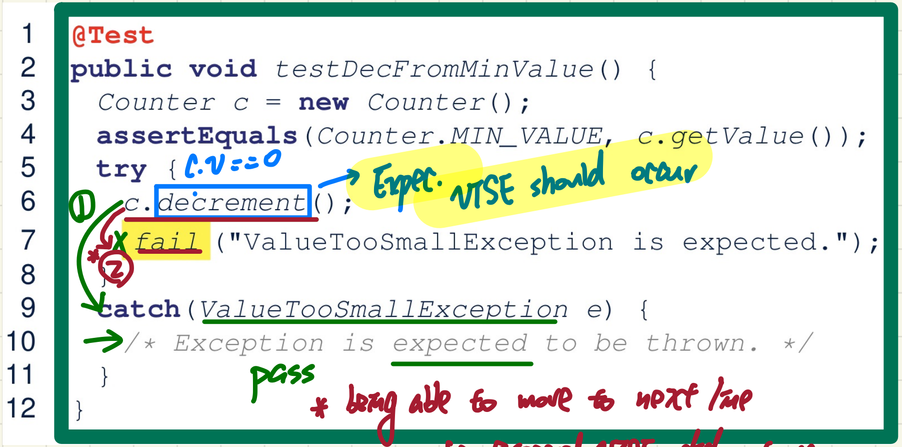
replace by VTSE

↳ Compilation error

① No VTLE occured → expected

→ test value is incremented

JUnit: An Exception Expected



```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. */
  }
}
```

c.v == 0

Expec. VTSE should occur

① ②

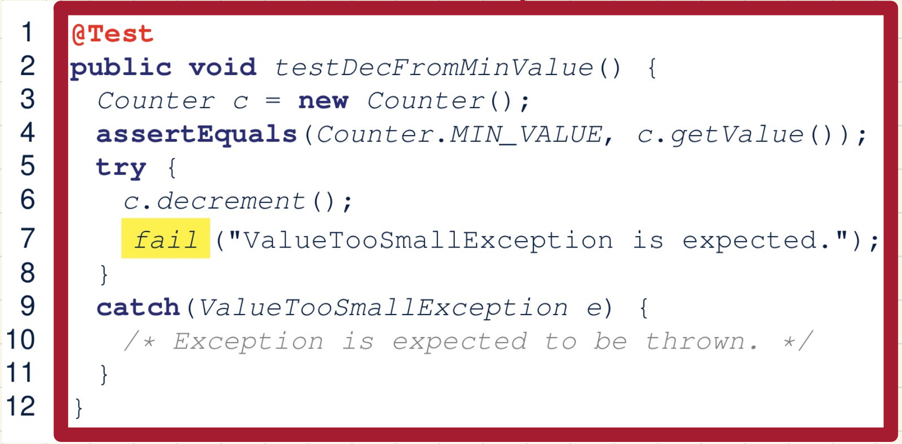
pass

* being able to move to next line means the expected VTSE did not occ.

What if increment is implemented correctly?

Expected Behaviour:

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

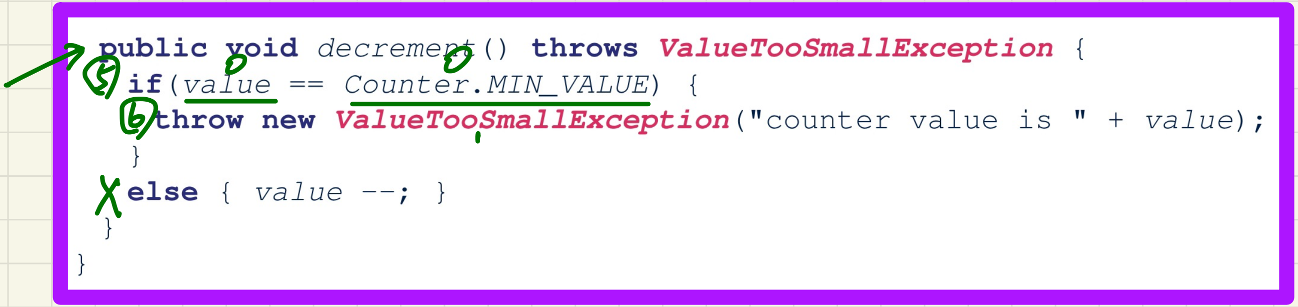


```java
@Test
public void testDecFromMinValue() {
  Counter c = new Counter();
  assertEquals(Counter.MIN_VALUE, c.getValue());
  try {
    c.decrement();
    fail("ValueTooSmallException is expected.");
  }
  catch(ValueTooSmallException e) {
    /* Exception is expected to be thrown. */
  }
}
```

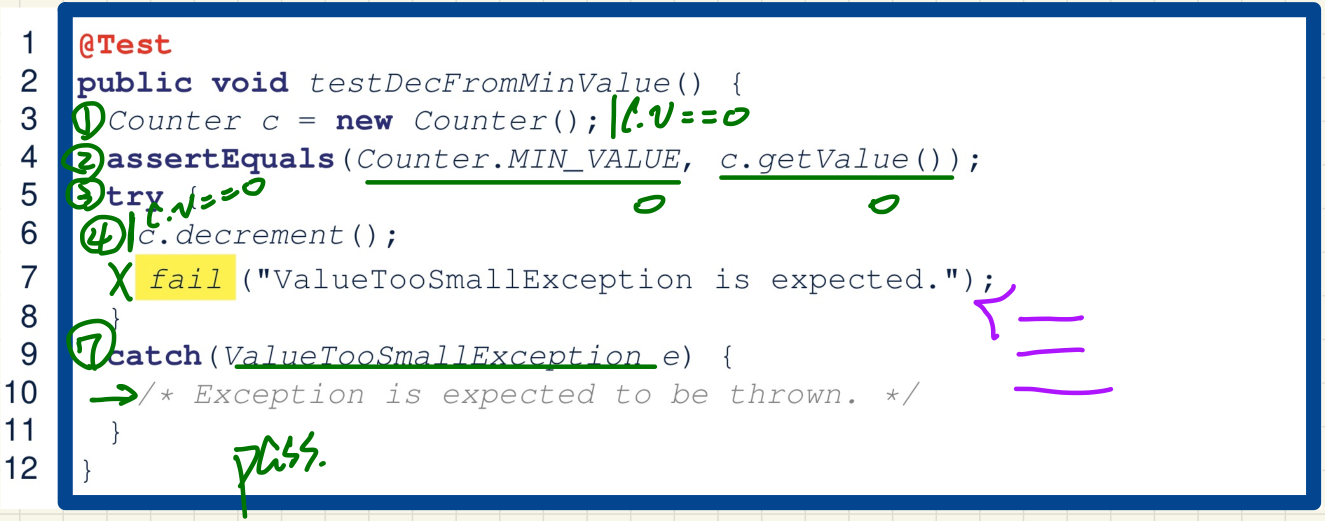
What if increment is implemented incorrectly?

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

Running JUnit Test 2 on Correct Implementation



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

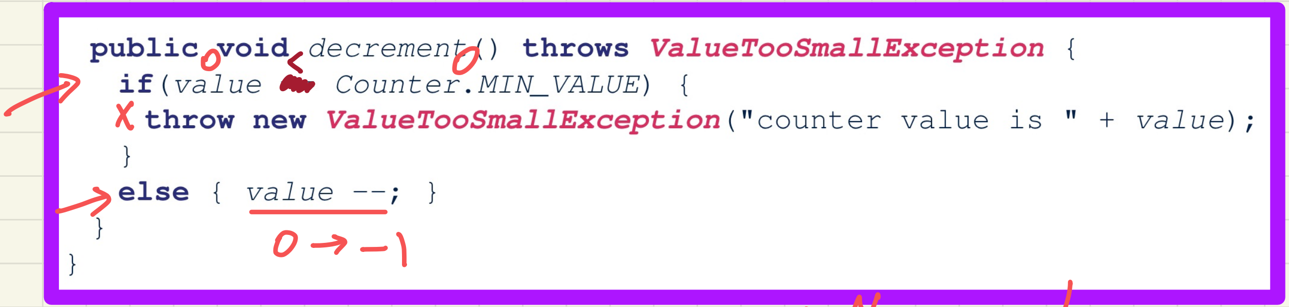


```java
@Test
public void testDecFromMinValue() {
 Counter c = new Counter();
 assertEquals(Counter.MIN_VALUE, c.getValue());
 try {
  c.decrement();
  fail ("ValueTooSmallException is expected.");
 }
 catch(ValueTooSmallException e) {
  /* Exception is expected to be thrown. */
 }
}
```

① c.v == 0
② 0 0
③ c.v == 0
④
⑤ 0 0
⑥
⑦
pass.

VTSE occurred
↳ expc flow disrupted
↳ pass !

Running JUnit Test 2 on Incorrect Implementation

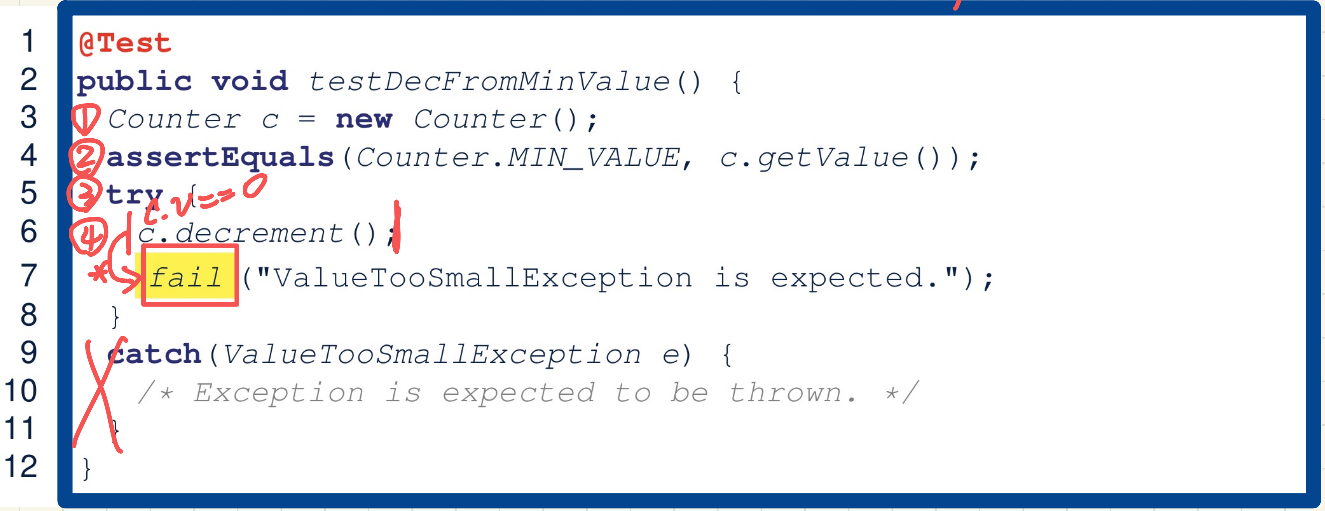


```java
public void decrement() throws ValueTooSmallException {
  if(value < Counter.MIN_VALUE) {
    throw new ValueTooSmallException("counter value is " + value);
  }
  else { value --; }
 }
}
```

0 0

0 → -1

* exp. flow normal ∵ no exception occurred.



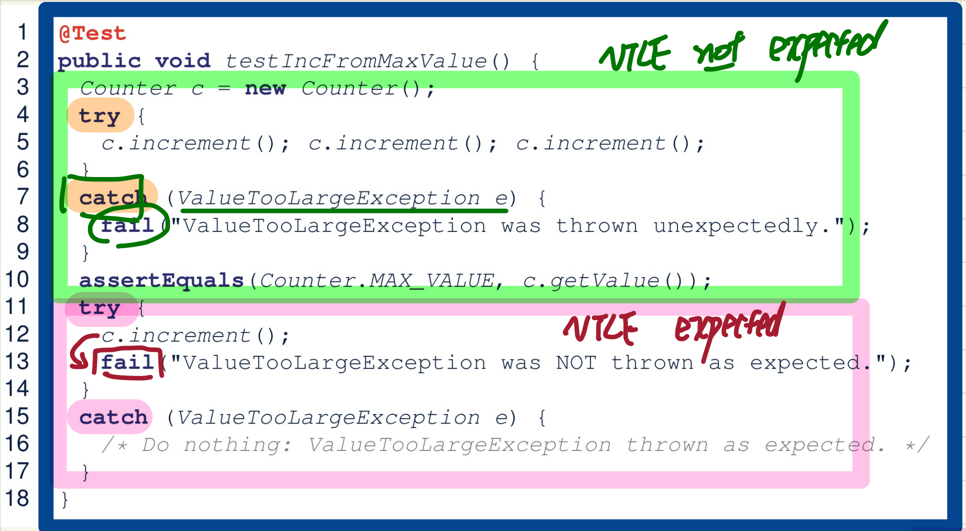
```java
@Test
public void testDecFromMinValue() {
  Counter c = new Counter();
  assertEquals(Counter.MIN_VALUE, c.getValue());
  try {
    c.decrement();
    fail("ValueTooSmallException is expected.");
  }
  catch(ValueTooSmallException e) {
    /* Exception is expected to be thrown. */
  }
}
```

① ② ③ ④

c.v == 0

*

JUnit: Exception Sometimes Expected, Somtimes Not

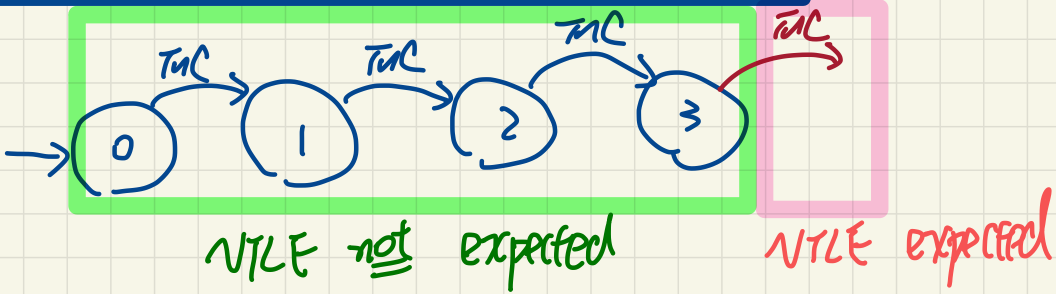


```java
@Test
public void testIncFromMaxValue() {
  Counter c = new Counter();
  try {
    c.increment(); c.increment(); c.increment();
  }
  catch (ValueTooLargeException e) {
    fail("ValueTooLargeException was thrown unexpectedly.");
  }
  assertEquals(Counter.MAX_VALUE, c.getValue());
  try {
 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

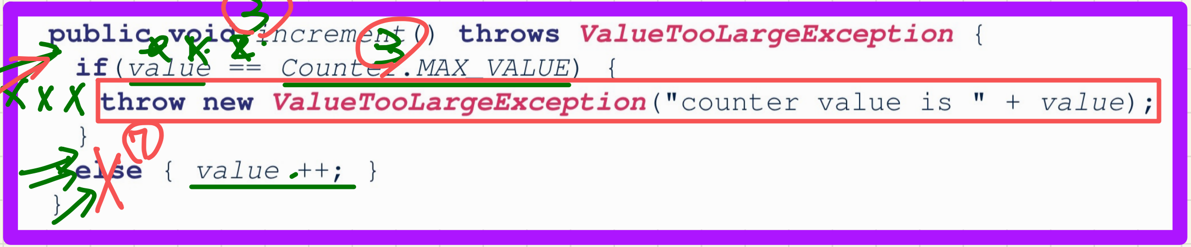
Expected Behaviour:

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



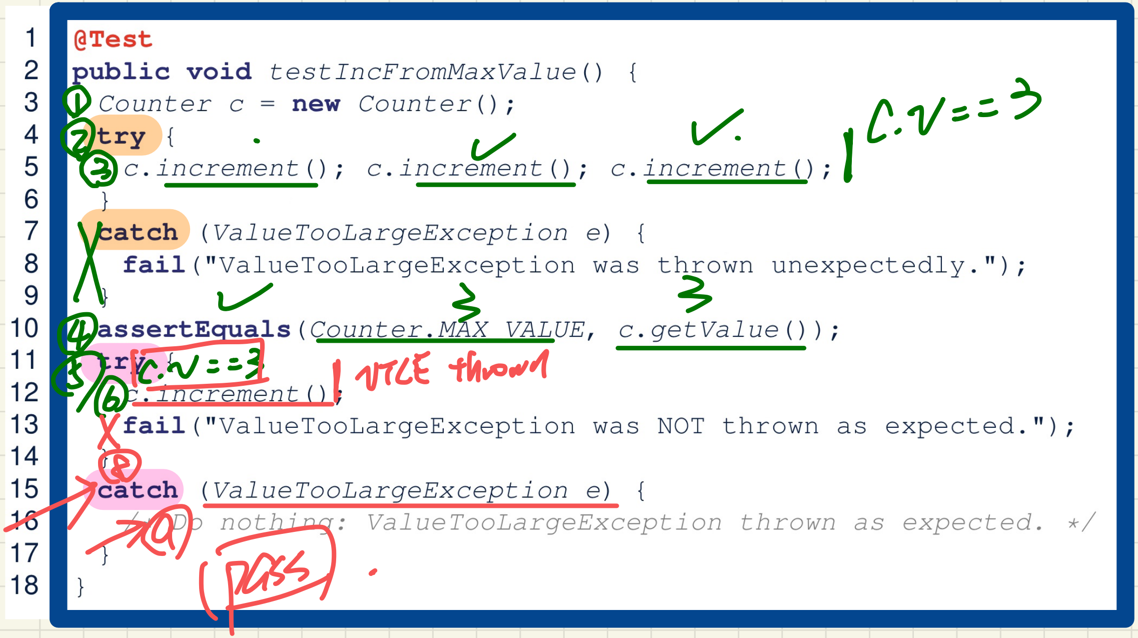
Running JUnit Test 3 on Correct Implementation



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

0 1 2 3

3



```
@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 == 3

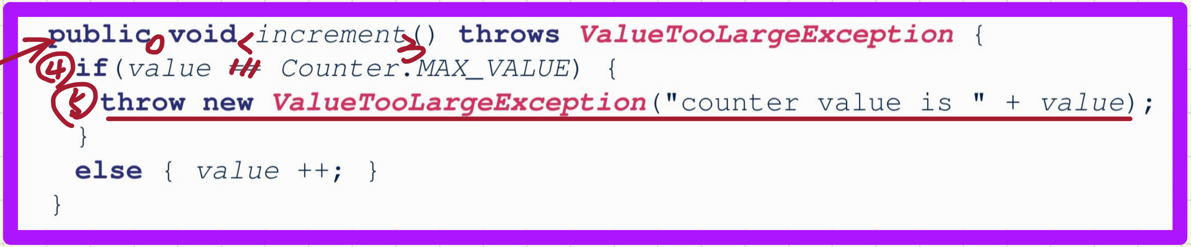
3 3

c.v == 3

VTLE thrown

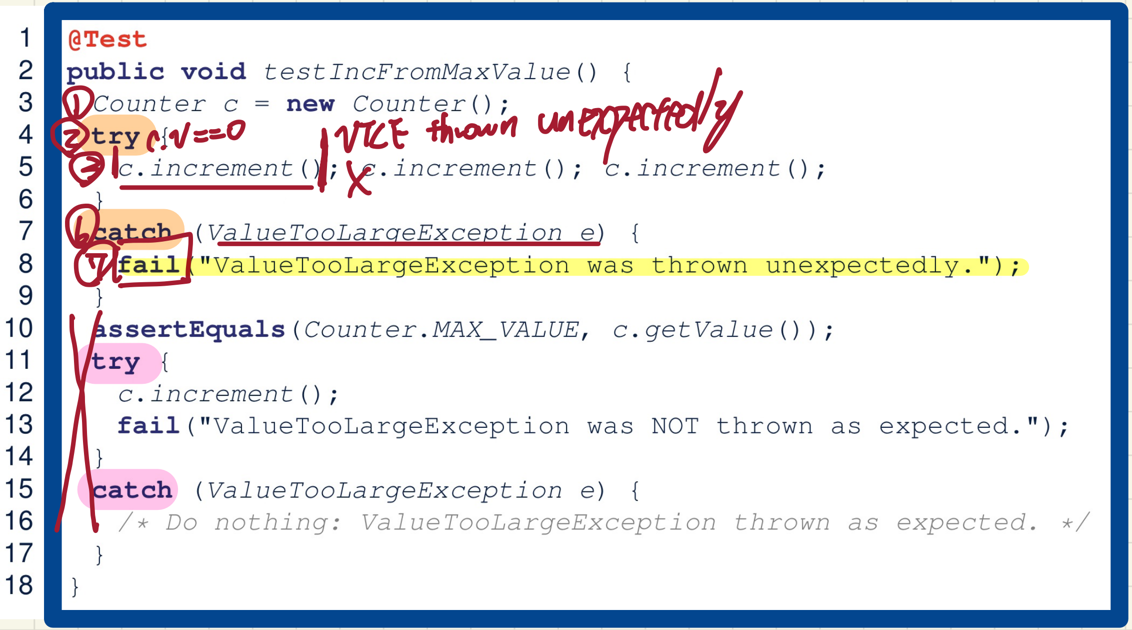
PASS

Running JUnit Test 3 on Incorrect Implementation



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

0 < 3

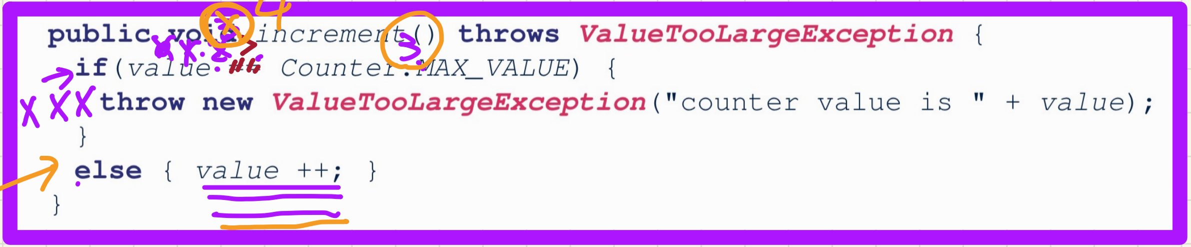


```java
@Test
public void testIncFromMaxValue() {
  Counter c = new Counter();
  try {
    c.increment(); c.increment(); c.increment();
  }
  catch (ValueTooLargeException e) {
    fail("ValueTooLargeException was thrown unexpectedly.");
  }
  assertEquals(Counter.MAX_VALUE, c.getValue());
  try {
 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

c.v==0

VTLE thrown unexpectedly

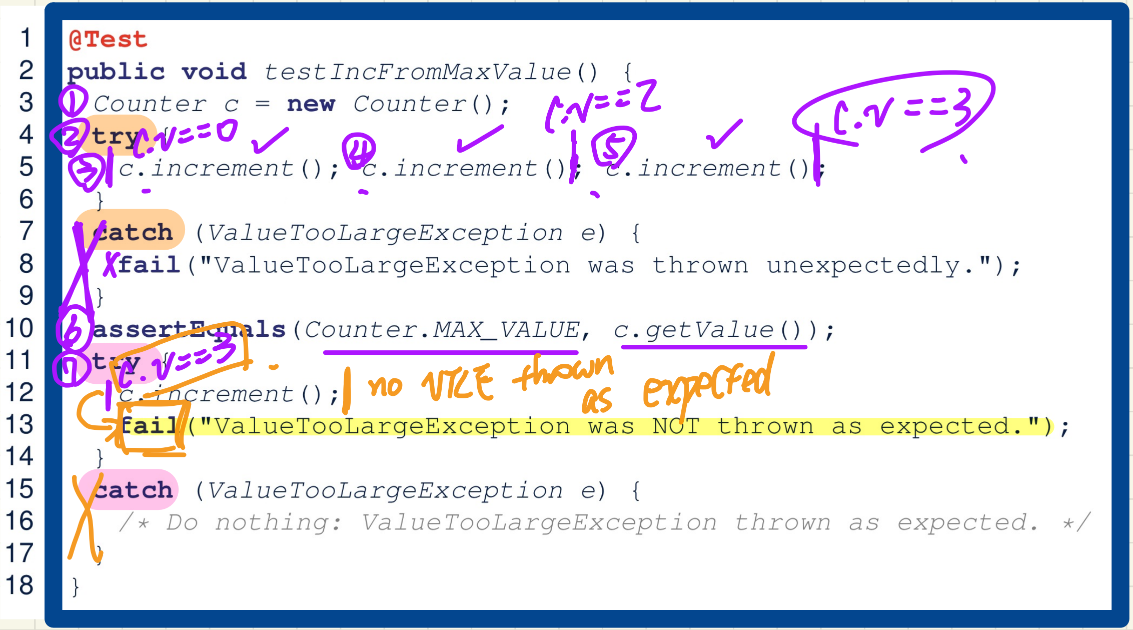
Running JUnit Test 3 on Incorrect Implementation



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

4

3



```java
@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

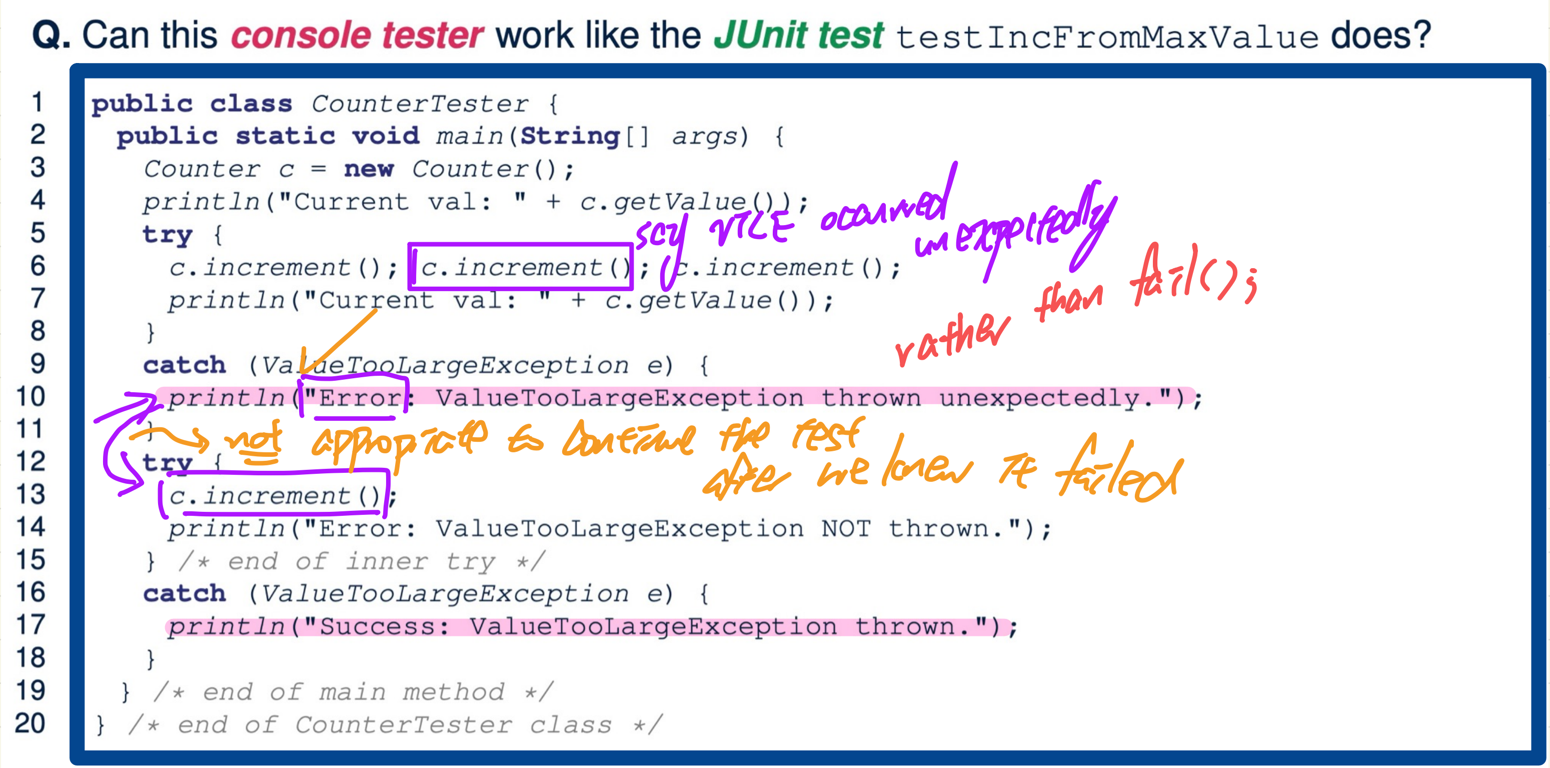
c.v==2

c.v==3

c.v==3

no VTLE thrown as expected

Exercise: Console Tester vs. JUnit Test



Q. Can this ***console tester*** work like the ***JUnit test*** `testIncFromMaxValue` does?

```
public class CounterTester {
  public static void main(String[] args) {
    Counter c = new Counter();
    println("Current val: " + c.getValue());
    try {
      c.increment(); c.increment(); c.increment();
      println("Current val: " + c.getValue());
    }
    catch (ValueTooLargeException e) {
      println("Error: ValueTooLargeException thrown unexpectedly.");
    }
    try {
      c.increment();
      println("Error: ValueTooLargeException NOT thrown.");
    } /* end of inner try */
    catch (ValueTooLargeException e) {
      println("Success: ValueTooLargeException thrown.");
    }
  } /* end of main method */
} /* end of CounterTester class */
```

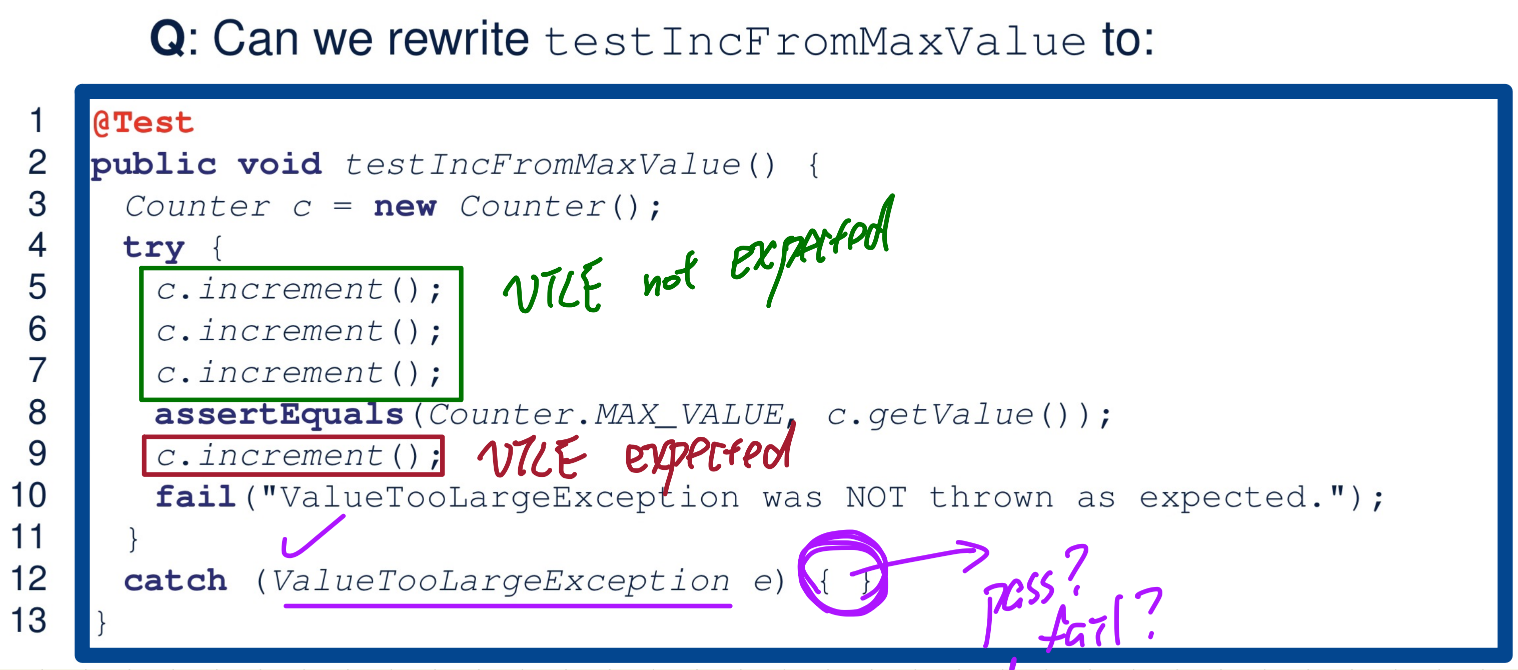
say VTLE occurred unexpectedly

rather than fail();

not appropriate to continue the test after we knew it failed

Hint: What if one of the first 3 c.increment() mistakenly throws a ValueTooLargeException?

Exercise: Combining catch Blocks?



Q: Can we rewrite `testIncFromMaxValue` to:

```
1  @Test
2  public void testIncFromMaxValue() {
3    Counter c = new Counter();
4    try {
5      c.increment();
6      c.increment();
7      c.increment();
8      assertEquals(Counter.MAX_VALUE, c.getValue());
9      c.increment();
10     fail("ValueTooLargeException was NOT thrown as expected.");
11   }
12   catch (ValueTooLargeException e) { }
13 }
```

VTLE not expected

VTLE expected

✓

pass?

fail?

VTLE occurred

Hint: Say Line 12 is executed,

is it clear if that ValueTooLargeException was thrown as expected?

Testing Many Values in a Single Test

Loops can make it effective on generating test cases:

```
@Test
public void testIncDecFromMiddleValues() {
  Counter c = new Counter();
  try {
    for(int i = Counter.MIN_VALUE; i < Counter.MAX_VALUE; i ++) {
      int currentValue = c.getValue();
      c.increment();
      assertEquals(currentValue + 1, c.getValue());
    }
    for(int i = Counter.MAX_VALUE; i > Counter.MIN_VALUE; i --) {
      int currentValue = c.getValue();
      c.decrement();
      assertEquals(currentValue - 1, c.getValue());
    }
  }
  catch(ValueTooLargeException e) {
    fail("ValueTooLargeException is thrown unexpectedly");
  }
  catch(ValueTooSmallException e) {
    fail("ValueTooSmallException is thrown unexpectedly");
  }
}
```

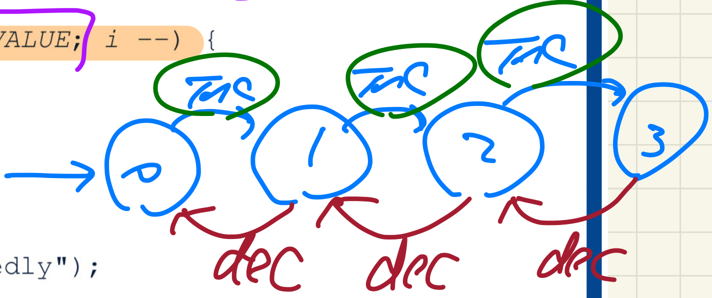
0 1 2] 3 times

→ invoked times

3

1

3 2 1] 3 times



inc
inc
inc
0
1
2
3
dec
dec
dec