

## Lecture 10 - Oct. 17

### Testing Exceptions & TDD

***Console Testers vs. JUnit Tests***  
***Regression Testing***

## Announcements

- Programming Test 1 Results: by the middle of next week
- Lab2 due this Friday
- Look ahead: WrittenTest2 & ProgTest2

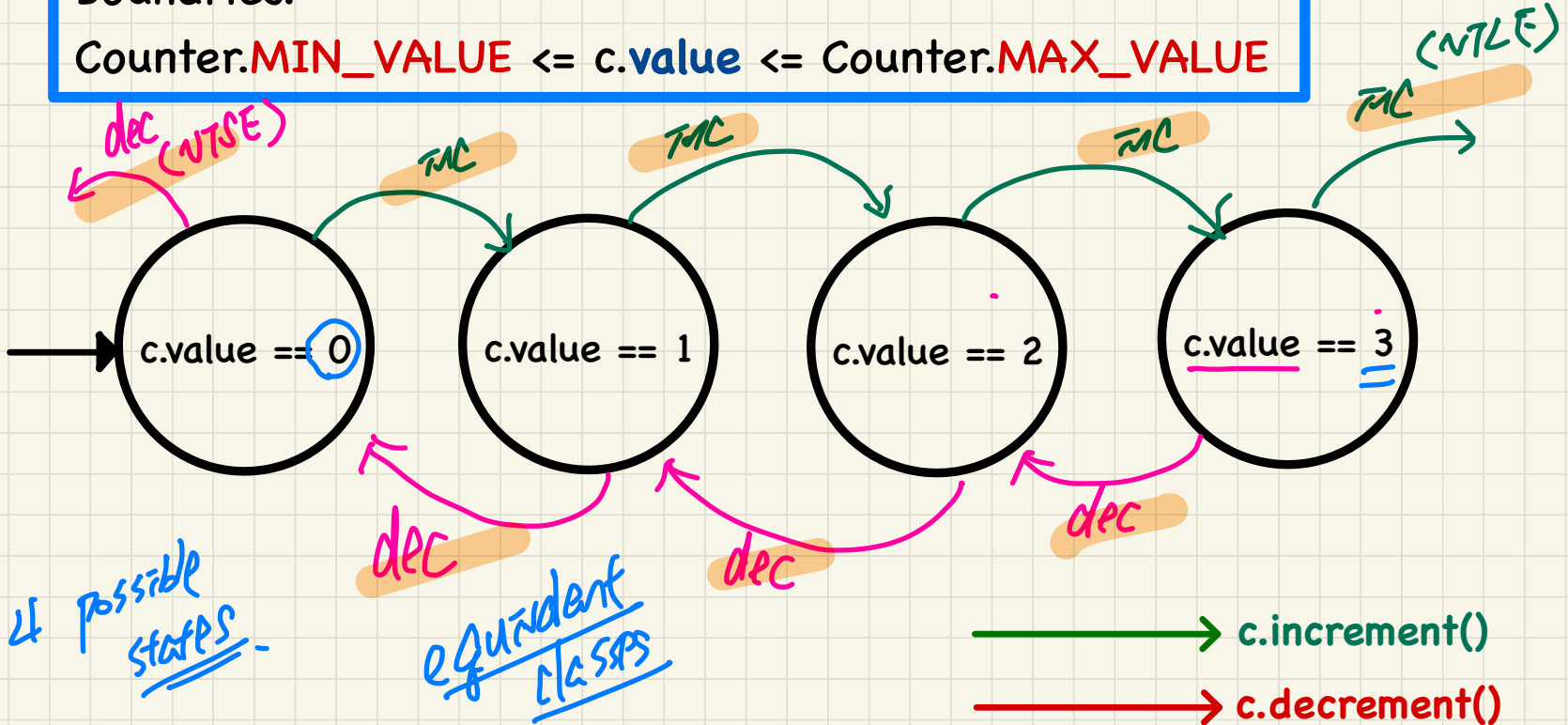
# Coming Up with Test Cases: A Single, Bounded Variable

state transition diagram

(EELS2001)

Boundries:

Counter.MIN\_VALUE <= c.value <= Counter.MAX\_VALUE



# 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; or
  - an **unexpected** exception (e.g., `NullPointerException`, `ArrayIndexOutOfBoundsException`) is thrown.
- **Success** if neither assertion failures nor **unexpected** exceptions occur.

*P.g. NullPointerException  
 ② AIOBE  
 test to fail:  
 (1) exception  
 (2) assertion failure.*

```

TestCounter.java
1 package tests;
2 import static org.junit.Assert.*;
3 import org.junit.Test;
4 public class TestCounter {
5     @Test
6     public void test() {
7         fail("Not yet implemented");
8     }
9 }
10
    
```

*Compare with:  
 system.out.println  
 ("Error: ...")*

*exer. flow not disrupted*

*disrupts exer. flow.*

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

# ① NullPointer Exception

correct object == null

\_\_\_\_\_ . m(...)  
not relevant

obj. m(...)  
null ⊗

obj. attr. attr? . m(...)

↓  
null  
⊕


null  
⊗

## ② IndexOutOfBoundsException

array a [n]

any integer expression  
p.g.  $\bar{c}$   
p.g.  $\bar{c}-1$

IOBE occurs if:

a →   
length is  $\bar{c}$

①  $n < 0$  ||  $n \geq a.length$

change this to  $||$ ?  
what if changing this to  $\&\&$ ? (Exercise)

② ! (  $0 \leq n$  &\&  $n < a.length$  )

# Examples: JUnit Assertions (1)

Consider the following class:

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

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

```
Point p;
assertNull(p); ✓ → null
assertTrue(p == null); ✓
assertFalse(p != null); ○ ✓
assertEquals(3, p.getX()); × /* NullPointerException */
p = new Point(3, 4);
assertNull(p); ■
assertTrue(p == null); ■
assertFalse(p != null); ■
assertEquals(3, p.getX()); ■
assertTrue(p.getX() == 3 && p.getY() == 4); ■
```

## 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**?

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

*Equals*



# JUnit: Where an **Exception** is **Not** Expected

```

1  @Test
2  public void testIncAfterCreation() {
3      Counter c = new Counter();
4      assertEquals(Counter.MIN_VALUE, c.getValue());
5      try {
6          c.increment(); → no exception thrown
7          assertEquals(1, c.getValue());
8      }
9      catch(ValueTooBigException e) {
10         /* Exception is not expected to be thrown. */
11         fail("ValueTooBigException is not expected.");
12     }
13 }

```

What if increment is implemented correctly?

```

1  @Test
2  public void testIncAfterCreation() {
3      Counter c = new Counter();
4      assertEquals(Counter.MIN_VALUE, c.getValue());
5      try {
6          c.increment(); → exception thrown unexpectedly
7          assertEquals(1, c.getValue());
8      }
9      catch(ValueTooBigException e) {
10         /* Exception is not expected to be thrown. */
11         fail("ValueTooBigException is not expected.");
12     }
13 }

```

What if increment is implemented incorrectly?

e.g., It only throws VTSE  
when c.value < 0

# JUnit: Where an Exception is Expected (1)

```

1  @Test
2  public void testDecFromMinValue() {
3  → Counter c = new Counter();
4  → assertEquals(Counter.MIN_VALUE, c.getValue());
5  try {
6  → c.decrement();
7  → X fail("ValueTooSmallException is expected.");
8  }
9  catch(ValueTooSmallException e) {
10  XX /* Exception is expected to be thrown. */
11  }
12  →

```

*→ throw VSE as expected*

## Console Tester

```

1  public class CounterTester1 {
2  public static void main(String[] args) {
3  Counter c = new Counter();
4  println("Init val: " + c.getValue());
5  try {
6  c.decrement();
7  println("Error: ValueTooSmallException NOT thrown.");
8  }
9  catch (ValueTooSmallException e) {
10  println("Success: ValueTooSmallException thrown.");
11  }
12  } /* end of main method */
13 } /* end of class CounterTester1 */

```

## JUnit Test

Scenario 1:  
dec implemented correctly

Scenario 2:  
dec not imp. correctly

```

1  @Test
2  public void testIncAfterCreation() {
3      Counter c = new Counter();
4      assertEquals(Counter.MIN_VALUE, c.getValue());
5      try {
6          c.increment();
7          assertEquals(1, c.getValue());
8      }
9      catch (ValueTooBigException e) {
10         /* Exception is not expected to be thrown. */
11         fail("ValueTooBigException is not expected.");
12     }
13 }

```

reaching this time means no exception happened unexpectedly.

NITBE happened unexpectedly.

```

1  @Test
2  public void testDecFromMinValue() {
3      Counter c = new Counter();
4      assertEquals(Counter.MIN_VALUE, c.getValue());
5      try {
6          c.decrement();
7          fail("ValueTooSmallException is expected.");
8      }
9      catch (ValueTooSmallException e) {
10         /* Exception is expected to be thrown. */
11     }
12 }

```

the expected exception did not occur

the expected exception occurred ⇒ pass

# JUnit: where an Exception is Expected (2.1)

## Console Tester

working

working

```

1  @Test
2  public void testIncFromMaxValue() {
3      Counter c = new Counter();
4      try {
5          c.increment(); c.increment(); c.increment();
6      }
7      catch (ValueTooLargeException e) {
8          fail("ValueTooLargeException was thrown unexpectedly.");
9      }
10     assertEquals(Counter.MAX_VALUE, c.getValue());
11     try {
12         c.increment();
13         fail("ValueTooLargeException was NOT thrown as expected.");
14     }
15     catch (ValueTooLargeException e) {
16         /* Do nothing: ValueTooLargeException thrown as expected. */
17     }
18 }

```

(unnested)

```

1  public class CounterTester2 {
2      public static void main(String[] args) {
3          Counter c = new Counter();
4          println("Current val: " + c.getValue());
5          try {
6              c.increment(); c.increment(); c.increment();
7              println("Current val: " + c.getValue());
8              try {
9                  c.increment();
10                 println("Error: ValueTooLargeException NOT thrown.");
11             } /* end of inner try */
12             catch (ValueTooLargeException e) {
13                 println("Success: ValueTooLargeException thrown.");
14             } /* end of inner catch */
15         } /* end of outer try */
16         catch (ValueTooLargeException e) {
17             println("Error: ValueTooLargeException thrown unexpectedly.");
18         } /* end of outer catch */
19     } /* end of main method */
20 } /* end of CounterTester2 class */

```

(nested)

## JUnit Test

# JUnit: where an Exception is Expected (2.2)

Recall the alternative to CounterTester2 that has **un-nested** try-catch blocks.

Why is the JUnit test logically correct but the **Console Tester** is not?

```

public class CounterTester2 {
    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 inner catch */
    } /* end of main method */
} /* end of CounterTester2 class */

```

→ NICE throw unexpectedly

→ un-nested ⇒ not working.

→ Console Tester

How not disrupted.

```

1 @Test
2 public void testIncFromMaxValue() {
3     Counter c = new Counter();
4     try {
5         c.increment(); c.increment(); c.increment();
6     }
7     catch (ValueTooLargeException e) {
8         fail("ValueTooLargeException was thrown unexpectedly.");
9     }
10    assertEquals(Counter.MAX_VALUE, c.getValue());
11    try {
12        c.increment();
13        fail("ValueTooLargeException was NOT thrown as expected.");
14    }
15    catch (ValueTooLargeException e) {
16        /* Do nothing: ValueTooLargeException thrown as expected. */
17    }
18 }

```

→ NICE throw unexpectedly

→ un-nested ⇒ working

→ test fails right away

→ JUnit Test

# Exercise

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 }

```

*if VTLCE thrown → it's unexpected*

*if VTLCE thrown → it's expected*

*it's not possible to know whether not the VTLCE is expected*

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:

```

1  @Test
2  public void testIncDecFromMiddleValues() {
3  → Counter c = new Counter();
4  try {
5      for(int i = Counter.MIN_VALUE; i < Counter.MAX_VALUE; i++) {
6          int currentValue = c.getValue();
7          c.increment(); → and unexpected VICE thrown here will fail the tests
8          assertEquals(currentValue + 1, c.getValue());
9      }
10     for(int i = Counter.MAX_VALUE; i > Counter.MIN_VALUE; i--) {
11         int currentValue = c.getValue();
12         c.decrement();
13         assertEquals(currentValue - 1, c.getValue());
14     }
15 }
16 catch (ValueTooLargeException e) { → MIN: 0
17     fail("ValueTooLargeException is thrown unexpectedly");
18 }
19 catch (ValueTooSmallException e) { → MAX: 1000
20     fail("ValueTooSmallException is thrown unexpectedly");
21 }
22 }

```

# Test-Driven Development (TDD): Regression Testing

