

## Lecture 10 - Oct. 8

### TDD with JUnit, Object Equality.

***JUnit Test: Exception Expected vs. Not  
Using Loops in JUnit Test Methods  
Default equals Method in Object Classes***

## Announcements/Reminders

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

# JUnit: An Exception Expected

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

*exp: VTSE should occur*

*fail*

*expected VTSE did not occur → fail*

*VTSE occurred as expected → pass*

What if increment is implemented correctly?

## Expected Behaviour:

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

```
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
```

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

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

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

abnormal flow  
→ pass

→ pass

# Running JUnit Test 2 on Incorrect Implementation



→

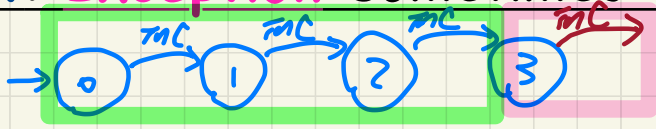
```
public void decrement() throws ValueTooSmallException {  
    ⑤ if (value ⑥ ≤ Counter.MIN_VALUE) {  
        X throw new ValueTooSmallException("counter value is " + value);  
    }  
    ⑦ else { value ⑧ --; }  
}
```

Handwritten notes: 0 → -1

normal flow  
→ test  
fails!

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

# JUnit: Exception Sometimes Expected, Sometimes Not



```
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 }
```

*NOTE not expected*

*NOTE 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++; }  
}
```

```
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 }
```

*Handwritten notes:*

- ①-⑤: Annotations for the first try block.
- ⑥: Annotation for the first catch block.
- ⑦: Annotation for the second try block.
- ⑧: Annotation for the second catch block.
- ⑨: Annotation for the assertEquals line.
- ⑩: Annotation for the second try block.
- ⑪: Annotation for the second catch block.
- ⑫: Annotation for the fail line in the second catch block.
- ⑬: Annotation for the do-nothing comment.
- ⑭: Annotation for the end of the test method.
- ⑮: Annotation for the assertEquals line.
- ⑯: Annotation for the fail line in the second catch block.
- ⑰: Annotation for the do-nothing comment.
- ⑱: Annotation for the end of the test method.
- ⑲: Annotation for the fail line in the second catch block.
- ⑳: Annotation for the do-nothing comment.
- ㉑: Annotation for the end of the test method.
- ㉒: Annotation for the fail line in the second catch block.
- ㉓: Annotation for the do-nothing comment.
- ㉔: Annotation for the end of the test method.
- ㉕: Annotation for the fail line in the second catch block.
- ㉖: Annotation for the do-nothing comment.
- ㉗: Annotation for the end of the test method.
- ㉘: Annotation for the fail line in the second catch block.
- ㉙: Annotation for the do-nothing comment.
- ㉚: Annotation for the end of the test method.
- ㉛: Annotation for the fail line in the second catch block.
- ㉜: Annotation for the do-nothing comment.
- ㉝: Annotation for the end of the test method.
- ㉞: Annotation for the fail line in the second catch block.
- ㉟: Annotation for the do-nothing comment.
- ㊱: Annotation for the end of the test method.
- ㊲: Annotation for the fail line in the second catch block.
- ㊳: Annotation for the do-nothing comment.
- ㊴: Annotation for the end of the test method.
- ㊵: Annotation for the fail line in the second catch block.
- ㊶: Annotation for the do-nothing comment.
- ㊷: Annotation for the end of the test method.
- ㊸: Annotation for the fail line in the second catch block.
- ㊹: Annotation for the do-nothing comment.
- ㊺: Annotation for the end of the test method.
- ㊻: Annotation for the fail line in the second catch block.
- ㊼: Annotation for the do-nothing comment.
- ㊽: Annotation for the end of the test method.
- ㊾: Annotation for the fail line in the second catch block.
- ㊿: Annotation for the do-nothing comment.

*Additional notes:*

- $c.v == 3$
- throws NTE as exp.
- pass!

# Running JUnit Test 3 on Incorrect Implementation

```
public void increment() throws ValueTooLargeException {  
    ④ if (value <= Counter.MAX_VALUE) {  
        ⑤ throw new ValueTooLargeException("counter value is " + value);  
    }  
    else { value++; }  
}
```

*unexpected/wrong*

```
1 @Test  
2 public void testIncFromMaxValue() {  
3     ① Counter c = new Counter();  
4     ② try {  
5         ③ c.increment(); c.increment(); c.increment();  
6     }  
7     ④ catch (ValueTooLargeException e) {  
8         ⑤ 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 }
```

*fail  
→ good test  
rejects  
a wrong  
imp.*



# Running JUnit Test 3 on Incorrect Implementation

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

```
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 }
```

test fails  
∴ the VLE  
did not  
occur as expected  
in 2nd phase

c.v == 3

VLE did not occur as expected

# Exercise: Console Tester vs. JUnit Test

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

```
1 public class CounterTester {
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         }
9         catch (ValueTooLargeException e) {
10            *println("Error: ValueTooLargeException thrown unexpectedly.");
11        }
12        try {
13            c.increment();
14            *println("Error: ValueTooLargeException NOT thrown.");
15        } /* end of inner try */
16        catch (ValueTooLargeException e) {
17            println("Success: ValueTooLargeException thrown.");
18        }
19    } /* end of main method */
20 } /* end of CounterTester class */
```

NOTE thrown unexpectedly

printing out an error about test failing will not stop from EXPR flow.

Error: keep tracing and see which msg printed

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 can be from here (as expected)*

*VTLE can be from here (expected)*

*Is a VTLE here expected or not expected?*

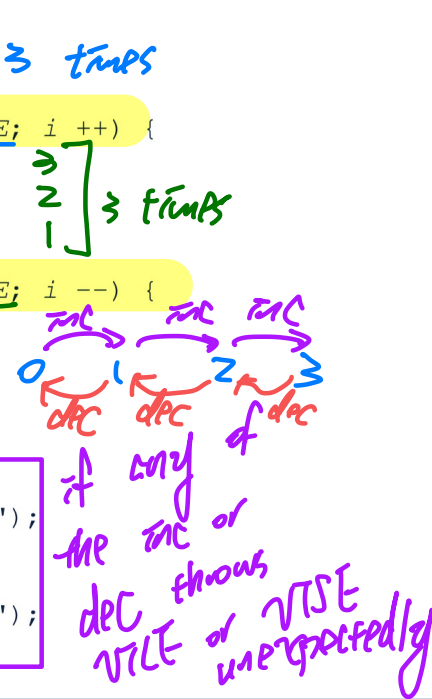
Hint: Say Line 12 is executed,

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();
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) {
17         fail("ValueTooLargeException is thrown unexpectedly");
18     }
19     catch (ValueTooSmallException e) {
20         fail("ValueTooSmallException is thrown unexpectedly");
21     }
22 }
```



\* p1.equals(p2) → this == obj.

```
class PointV1 {
```

```
    x
```

```
    y
```

```
    Point() { ... }
```

\* no equals method explicitly declared \*/

```
}
```

```
PointV1  
PointV1
```

```
p1 = ... *  
p2 = ..
```

```
p1.equals(p2)
```

no compilation error

↓ default version Object class

# The equals Method: To **Override** or **Not**?

```
public class Object {
```

```
...
```

```
public boolean equals(Object obj) {
```

```
    return this == obj;
```

```
    }
```

*default imp: ref. equality -*

extends

extends

*equals method  
overridden /  
redefined*

```
public class PointV1 {
```

```
    private double x;
```

```
    private double y;
```

```
    public PointV1 (double x, double y) {
```

```
        this.x = x;
```

```
        this.y = y;
```

```
    }
```

*equals not explicitly  
declared*

*→ use the default  
from Object class*

```
public class PointV2 {
```

```
    private int x; private int y;
```

```
    public PointV2 (int x, int y) { ... }
```

```
    public boolean equals(Object obj) {
```

```
        if (this == obj) { return true; }
```

```
        if (obj == null) { return false; }
```

```
        if (this.getClass() != obj.getClass()) { return false }
```

```
        Point other = (PointV2) obj;
```

```
        return this.x == other.x
```

```
            && this.y == other.y;
```

```
    }
```