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      fail("ValueTooSmallException is expected.");
8    }
9    catch(ValueTooSmallException e) {
10     /* Exception is expected to be thrown. */
11   }
12 }
```

exp: VTSE should occur

c.v == 0

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

```
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 }
```

abnormal flow → pass

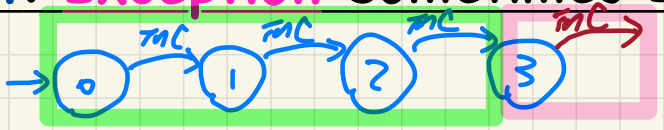
Running JUnit Test 2 on Incorrect Implementation

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

```
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 }
```

normal flow → test fails!

JUnit: Exception Sometimes Expected, Somtimes 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 }
```

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++; }
}
```

```
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 }
```

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 ++; }
}
```

unexpected/wrong

```
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 }
```

fail :(

good test reports a wrong impl.

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 VTLE did not occur, as expected in 2nd phase

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 */
```

NTLE thrown unexpected

printing out an error about test failing will not stop the exception flow.

Exercise: keep tracing until 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 (an expected)

VTLE can be from here (expected)

Is a VTLE here expected or not expected?

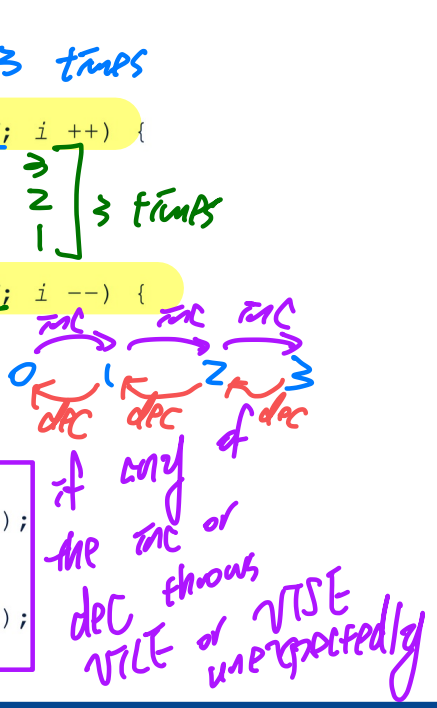
Hint: Say Line 12 is executed,

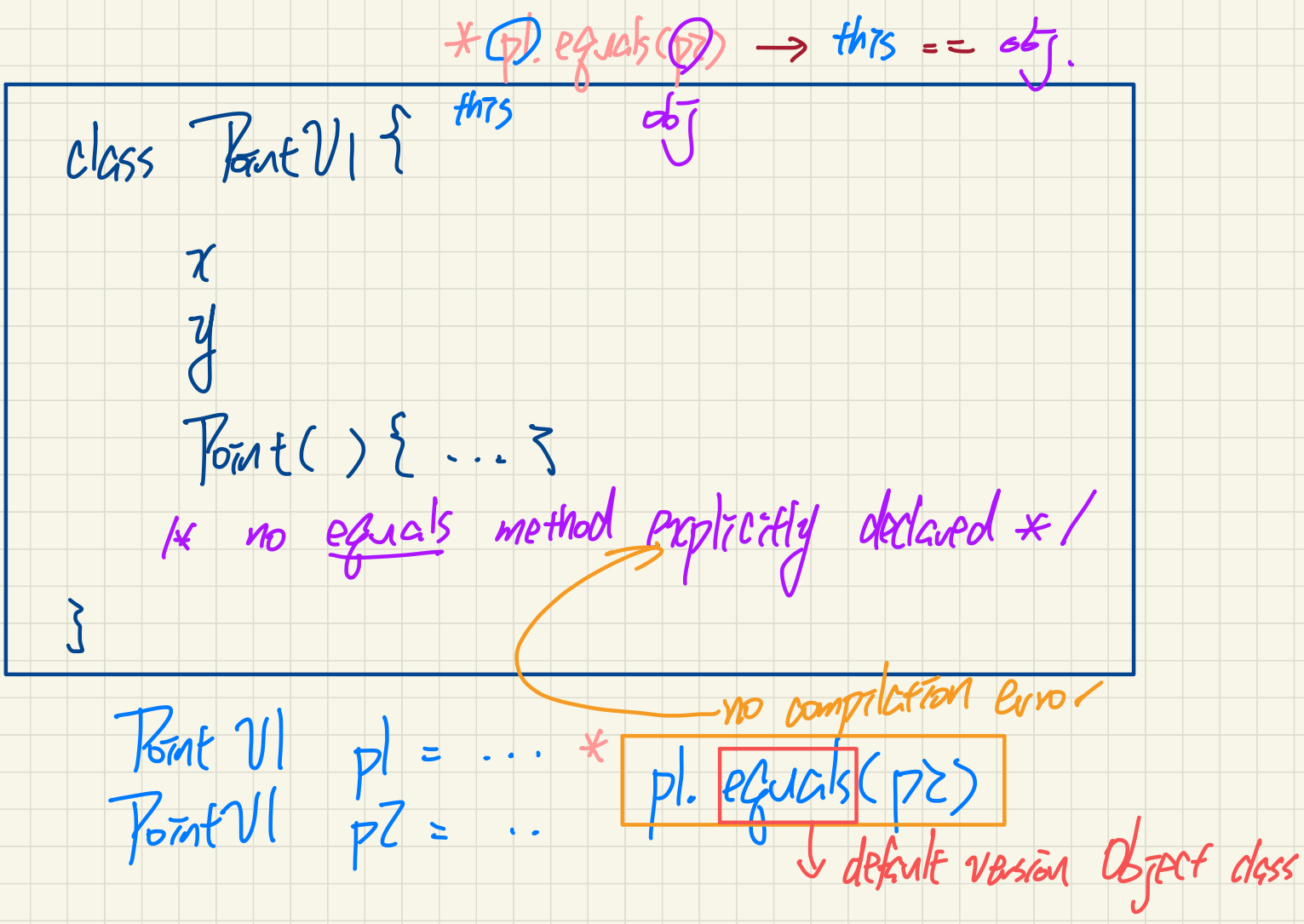
is it clear if that ValueTooLargeException was thrown as expected?

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.
this
obj
class PointV1 {
x
y
Point( ) { ... }
/* no equals method explicitly declared */
}
no compilation error
PointV1 p1 = ... *
PointV1 p2 = ..
p1.equals(p2)
default version Object class

The equals Method: To Override or Not?