Lecture 2b

Part A

Test-Driven Development (TDD) - Counter Problem, Review on Exceptions

Review: Specify-or-Catch Principle

Approach 1 – Specify: Indicate in the method signature that a specific exception might be thrown.

Example 1: Method that throws the exception

specify opt.

```
class C1 {
  void m1(int x) throws ValueTooSmallException {
    if(x < 0) {
      throw new ValueTooSmallException("val " + x);
    }
  }
}
```

↳ handle an error

↳ where the exception is originated

Example 2: Method that calls another which throws the exception

```
class C2 {
  C1 c1;
  void m2(int x) throws ValueTooSmallException {
    c1.m1(x);
  }
}
```

specify opt.

c1.m1(x); → may throw a UTSE → subject to catch-or-specify req.

C1.m1

Review: Specify-or-Catch Principle

Approach 2 – Catch: Handle the thrown exception(s) in a try-catch block.

```
class C3 {
  public static void main(String[] args) {
    Scanner input = new Scanner(System.in);
    int x = input.nextInt();
    C2 c2 = new c2();
    try {
      c2.m2(x);
    }
    catch(ValueTooSmallException e) { ... }
  }
}
```

C.O.

may throw VTSE → must either catch or specify.

Exercise: Put VTSE instead

match one of the exceptions that might come from the catch block

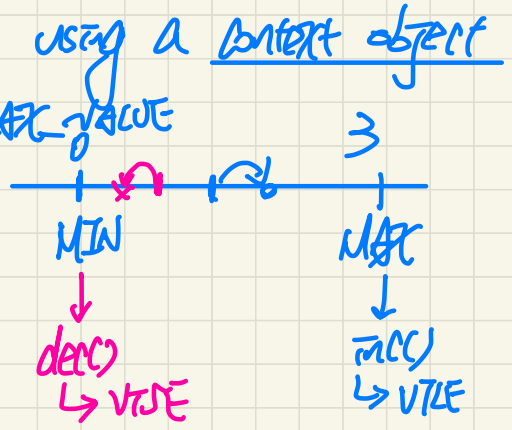
VTSE will not be propagated further.

A Class for Bounded Counters

```
public class Counter {
  public final static int MAX_VALUE = 3;
  public final static int MIN_VALUE = 0;
  private int value;
  public Counter() {
    this.value = Counter.MIN_VALUE;
  }
  public int getValue() {
    return value;
  }
  ... /* more later! */
```

no need to access them using a Context object

↳ Counter.MAX_VALUE




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

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

specify opt.

Exercises:

* ① value < Counter.MAX_VALUE

② value > Counter.MAX_VALUE

** ① value < Counter.MIN_VALUE

② value > Counter.MIN_VALUE

Lecture 2b

Part B

Test-Driven Development (TDD) - Manual, Console Testers

Manual Tester 1 from the Console

```
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 */
```

correct → VTSE thrown as expected

What if decrement is implemented correctly?

Expected Behaviour:

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

```
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 */
```

incorrect → VTSE not thrown

What if decrement is implemented incorrectly?

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

Running Console Tester 1 on Correct Implementation

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

thrown as expected

```
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 */
```

Running Console Tester 1 on Incorrect Implementation

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

```
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 */
```

Manual Tester 2 from the Console

Test Case 3

- Nothing unexpected occurs.
- Everything expected occurs.

Test Case 1

VTLE thrown unexpectedly

Test Case 2

VTLE not thrown as expected

```
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 */
```

expected VTLE

not thrown

unexpectedly thrown VTLE

Running Console Tester 2 on (Correct) Implementation 1

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

```
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 */
```

As soon as the VTLE is handled, it won't be propagated further

Running Console Tester 2 on (Incorrect) Implementation 2

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

```
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 */
```

Running Console Tester 2 on (Incorrect) Implementation 3

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

```
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 */
```

Exercise

Question. Can this alternative to `ConsoleTester2` work (without nested `try-catch`)?

```
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      }
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     } /* end of inner catch */
19   } /* end of main method */
20 } /* end of CounterTester2 class */
```

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

A Manual, Iterative Console Tester

```
import java.util.Scanner;
public class CounterTester3 {
  public static void main(String[] args) {
    Scanner input = new Scanner(System.in);
    String cmd = null;  Counter c  = new Counter();
    boolean userWantsToContinue = true;
    while(userWantsToContinue) {
      println("Enter \"inc\", \"dec\", or \"val\":");
      cmd = input.nextLine();
    try {
       if(cmd.equals("inc")) {  c.increment() ; }
       else if(cmd.equals("dec")) {  c.decrement() ; }
       else if(cmd.equals("val")) { println( c.getValue() ); }
       else { userWantsToContinue = false; println("Bye!"); }
      } /* end of try */
      catch(ValueTooLargeException e){ println("Value too big!"); }
      catch(ValueTooSmallException e){ println("Value too small!"); }
    } /* end of while */
  } /* end of main method */
} /* end of class CounterTester3 */
```

Lecture 2b

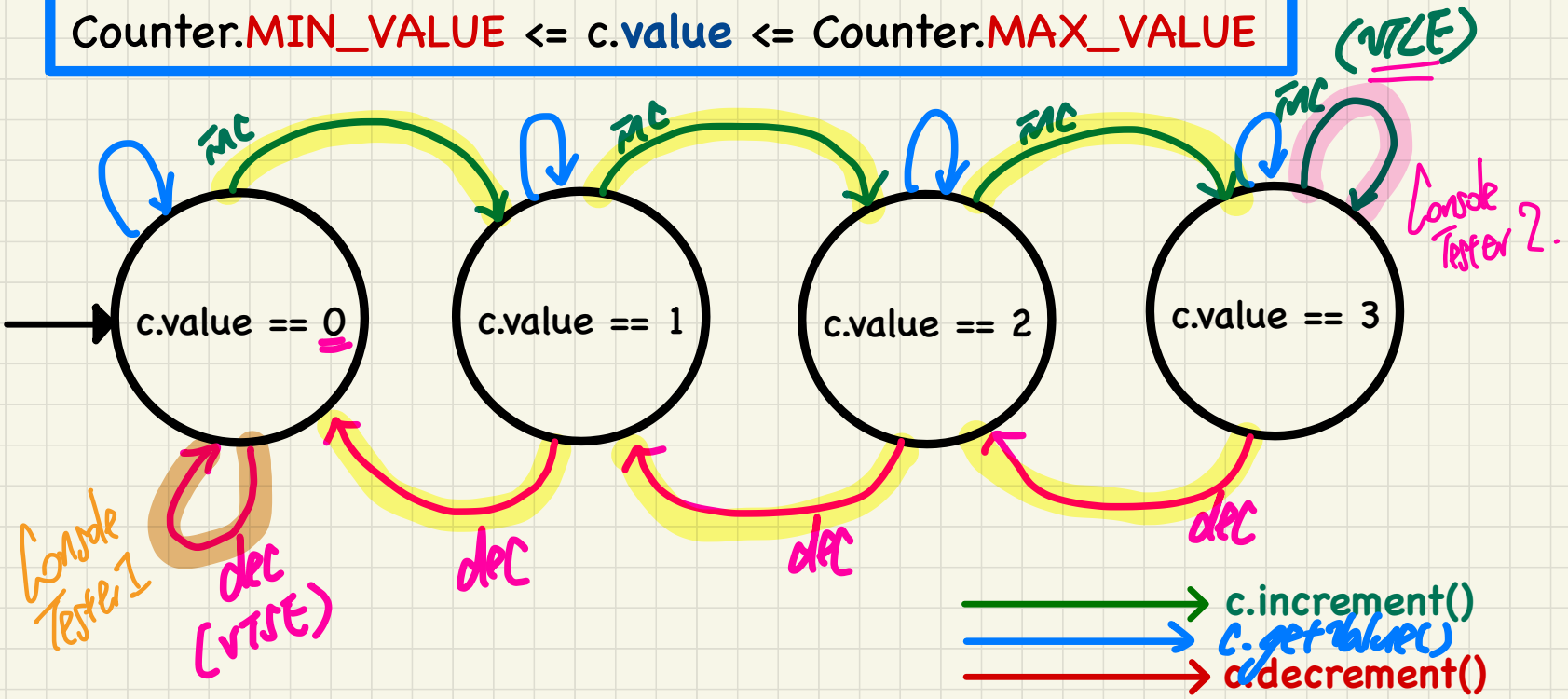
Part C

Test-Driven Development (TDD) - Test Cases for a Bounded Variable

Coming Up with Test Cases: A Single, Bounded Variable

Boundries:

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



Lecture 2b

Part D

Test-Driven Development (TDD) - JUnit Testing via Assertions

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`, `ArrayIndexOutOfBoundException`) is thrown.
- ***Success*** if neither assertion failures nor *unexpected* exceptions occur.

TestCounter.java

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

do nothing.

② No assertions run to check expected vs. actual values.

useless

①: no meaningful manipulation of objects instantiated from classes in the model package.

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

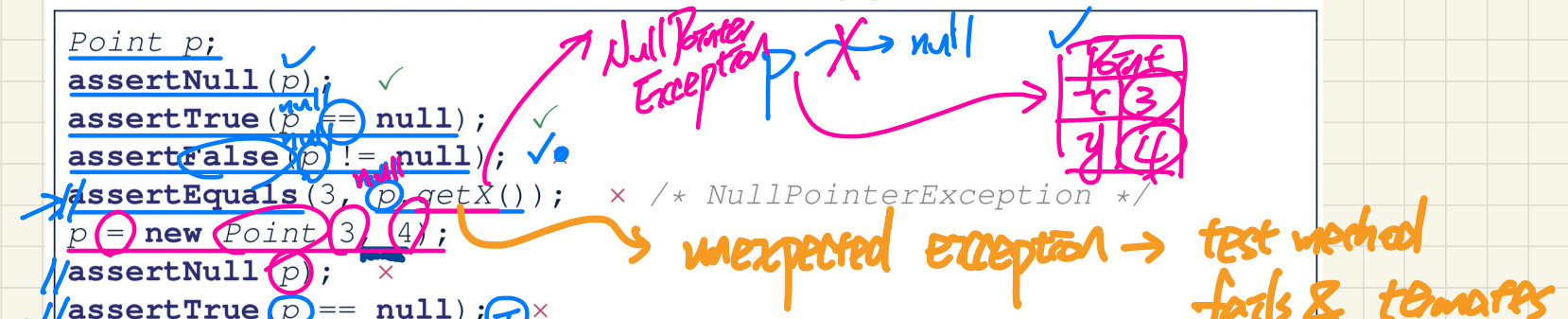
Examples: JUnit Assertions (1)

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);                                  ✓
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); ✓
```

tolerance

Equals ↳ expected ↳ actual

c ↝ Circle | radius 3.4

Lecture 2b

Part E

Test-Driven Development (TDD) - Automated, JUnit Test Cases

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();
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 }
```

automated.

VTLE not expected

counterpart in Console Tester: println(c.getValue());

examine manually that 0 is printed to console

reaching this line means VTLE thrown unexpectedly.

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();
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 decrement is implemented incorrectly?

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

JUnit: Where an Exception is Expected (1)

JUnit Test

```
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 }
```

doing nothing is the most natural way for passing a test

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: where an Exception is Expected (2)

Console Tester

```
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 */
```

```
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 → unexpected

fail → VTLE not thrown as expected.

test passes

JUnit Test

if this assertion is executed, the entire test method will terminate and fail.

Exercise

Why is the JUnit test logically correct but the Console Tester is 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. */
  }
}
```

reaching this line will cause the rest of the test method to be bypassed and fail.

logically correct

```java
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 */
```

executing this line will not prevent the rest of the method from being executed

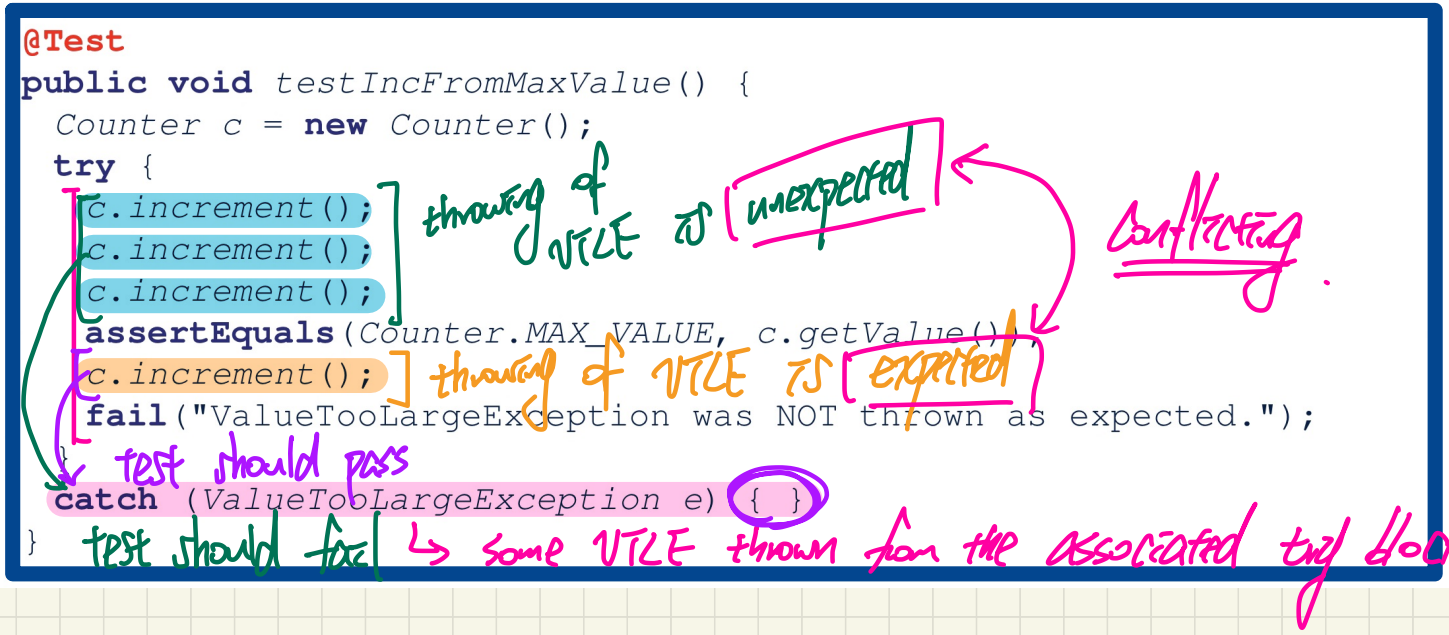
logically incorrect

inappropriate if there was already an error.

Exercise

Q: Can we rewrite `testIncFromMaxValue` to:

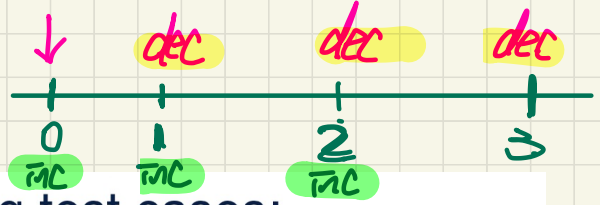
```
1  @Test
2  public void testIncFromMaxValue() {
3    Counter c = new Counter();
4    try {
5      c.increment();
6      c.increment();
7      c.increment();
8      assertEquals(Counter.MAX_VALUE, c.getValue());
9      c.increment();
10     fail("ValueTooLargeException was NOT thrown as expected.");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



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 }
```

0 1 2

3 iterations

3 2 1

→ No exception (else or else if) is expected

Lecture 2b

Part F

Test-Driven Development (TDD) - Regression Testing

Test-Driven Development (TDD): Regression Testing