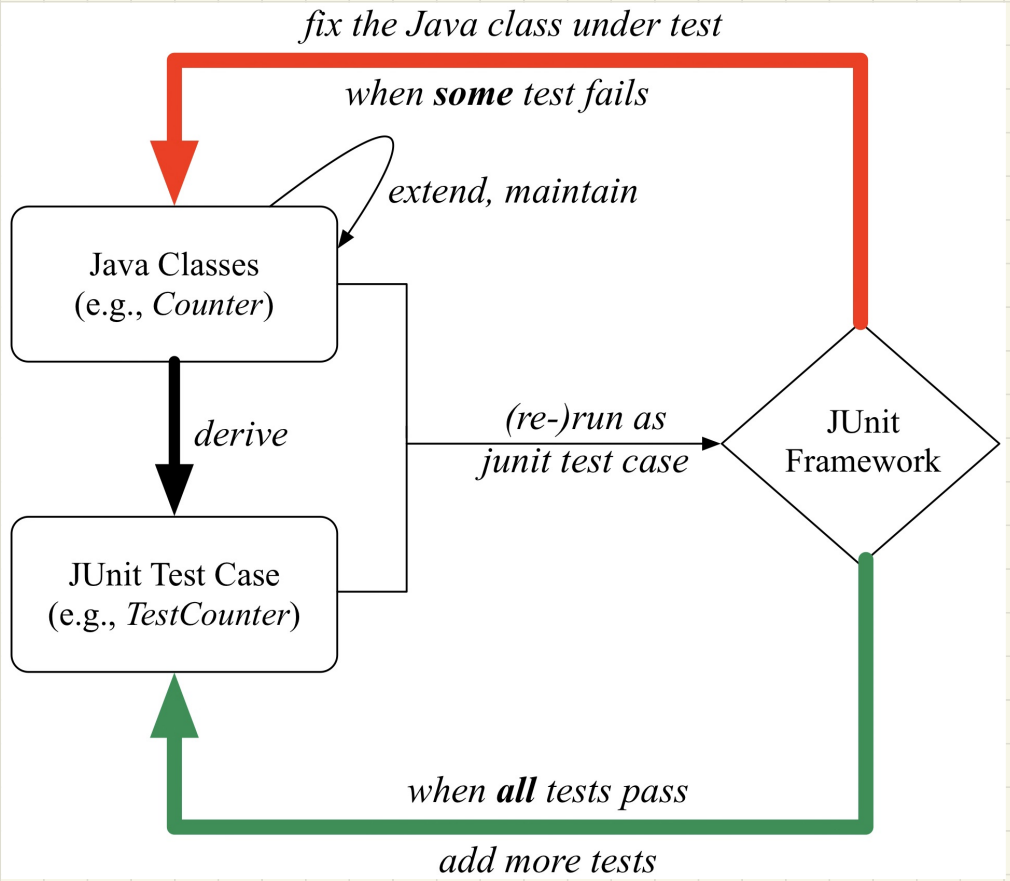
JUnit Test Method vs. Method Under Test

```
@Test
public void test() {
  MyClsss o = new MyClass();
  assertEquals(23, o.getValue());
}
```

Test-Driven Development (TDD): Regression Testing



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.

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
```

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);
assertTrue(p == null);
assertFalse(p != null);
assertEquals(3, p.getX());
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);
assertEquals(36.2984, c.getArea(), 0.01);
```

JUnit: An Exception Not Expected

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

What if increment is implemented correctly?

Expected Behaviour:

Calling c.increment() when c.value is 0 should **not** trigger a ValueTooLargeException

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

What if increment is implemented incorrectly?

e.g., It throws VTLE when c.value < Counter.MAX_VALUE

Running JUnit Test 1 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 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(ValueTooLargeException e) {
10     /* Exception is not expected to be thrown. */
11     fail("ValueTooLargeException is not expected.");
12   }
13 }
```

Running JUnit Test 1 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 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(ValueTooLargeException e) {
10     /* Exception is not expected to be thrown. */
11     fail("ValueTooLargeException is not expected.");
12   }
13 }
```

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 }
```

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 }
```

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 }
```

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 ++; }
}
```

```
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 ++; }
}
```

```
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 ++; }
}
```

```
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 }
```

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 */
```

Hint:

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 }
```

Hint: [redacted]

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 }
```