Lecture 9 - Oct. 3

TDD with JUnit

Deriving Test Cases
JUnit Test Method vs. Method Under Test
Regression Testing
JUnit Test: An Exception Not Expected

Announcements/Reminders

- **Written Test 1** result to be released Fri or Mon
- **Lab1** due tomorrow (Friday) at **noon**
- **Lab2** to be released tomorrow

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

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

origin of excep.

Example 2: Method that calls another which throws the exception

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

caller of some method that might throw exception.

Review: Specify-or-Catch Principle

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

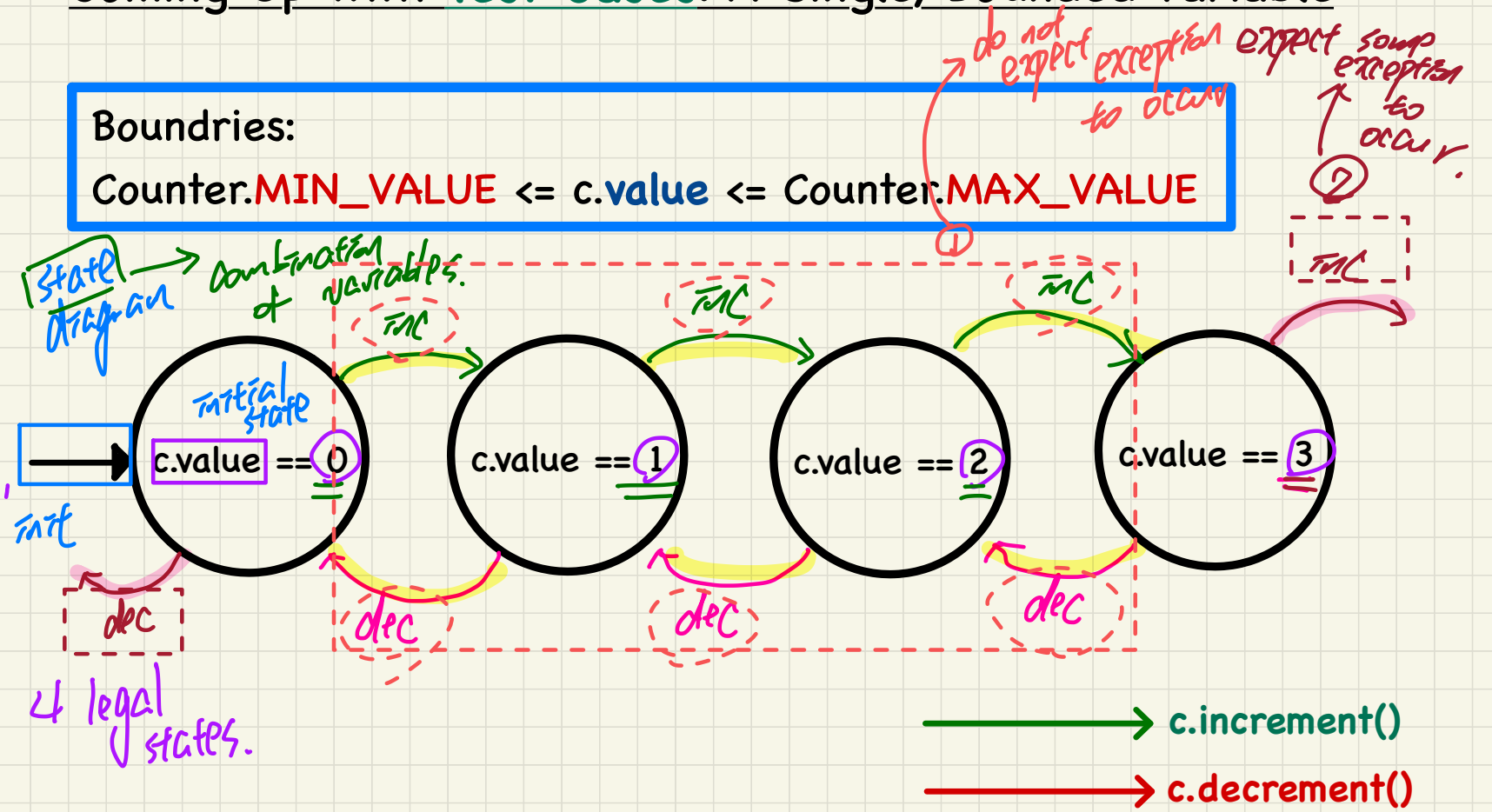
```
class C3 {
  public static void main(String[] args) {
    Scanner input = new Scanner(System.in);
    int x = input.nextInt();
    C2 c2 = new c2();
    try {
      c2.m2(x);
    }
    catch(ValueTooSmallException e) { ... }
  }
}
```

caller

Coming Up with Test Cases: A Single, Bounded Variable

Boundries:

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



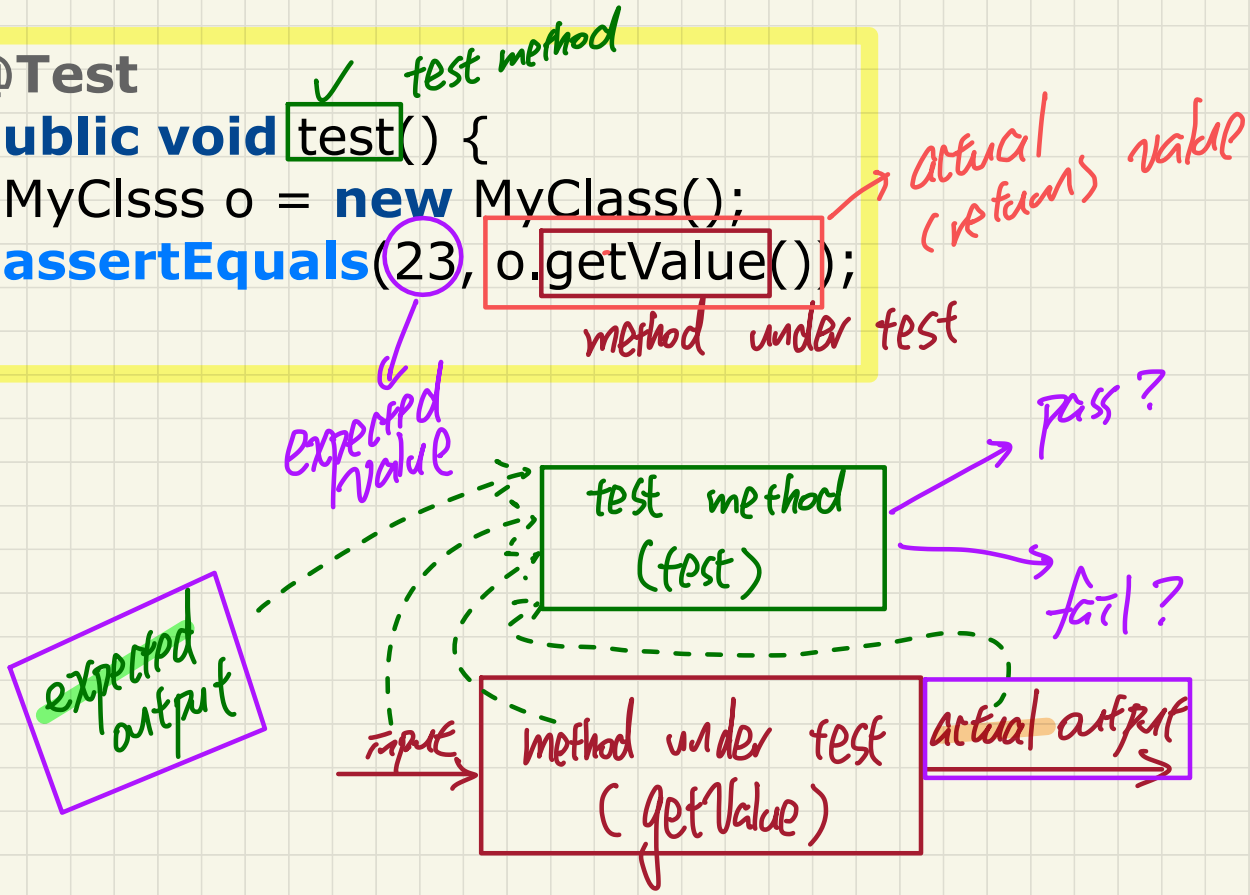
A Class for Bounded Counters

```
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!
```

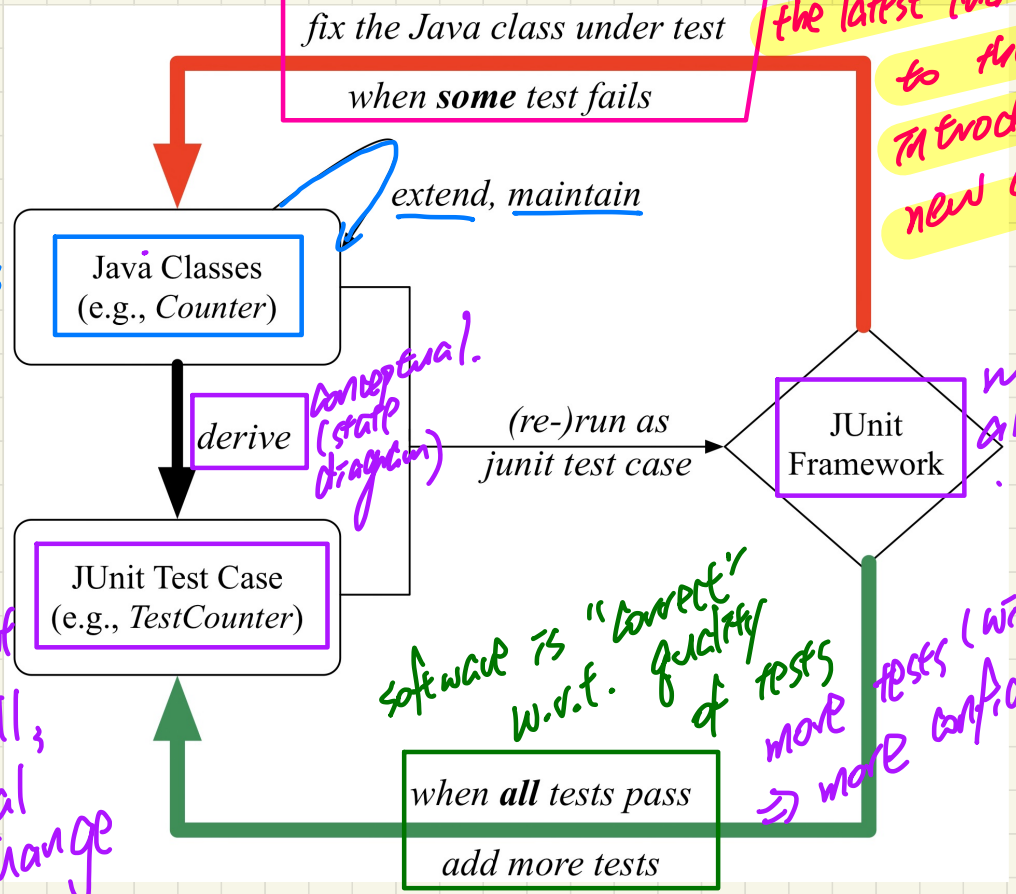
```
/* class Counter */
  public void increment() throws ValueTooLargeException {
    if(value == Counter.MAX_VALUE) {
      throw new ValueTooLargeException("counter value is " + value);
    }
    else { value ++; }
  }

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

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



the latest change made to the devel. code introduces a new error

classes under test

conceptual (state diagram)

run all tests

frequency of re-running all tests is important

Ideal: every small, logical change

software is "correct" w.r.t. quality of tests

more tests (with quality) ⇒ more confidence

A Default Test Case that Fails

fail();

The result of running a test is considered:

- ***Failure*** if either
  - an ***assertion failure*** (e.g., caused by `fail`, `assertTrue`, `assertEquals`) occurs
  - an unexpected (e.g., *`NullPointerException`*, *`ArrayIndexOutOfBoundException`*) ~~or unhandled~~ ***exception*** thrown
- ***Success*** if neither ***assertion failures*** nor (unexpected ~~or unhandled~~) ***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
```

assertTrue(false);
assertFalse(true);

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);
```

expected actual tolerance (ε)

=

expected − ε ≤ c.getArea() ≤ expected + ε

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

may throw VTLE

reaching this line means no exception occurred.

reaching the catch block means VTLE occurred

What if increment is implemented correctly?

v1 of TMC. (no VTLE thrown)

Expected Behaviour:

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

```
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 }
```

What if increment is implemented incorrectly?

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

v2 of TMC (VTLE thrown unexpectedly)

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 ++; }
}
```

Correct imp.

prediction of test result: pass

```
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 ++; }
}
```

wrong impl. ⇒ test should fail

prediction & test result: fail ✓

```
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 }
```