Lecture 7

Wednesday September 25

- WRITTEN TEST REVIEW SESSION

CLH M 10am ~ 12noon

MONDAY SEPTEMBER 30

POST YOUR QUESTIONS ON A GOOGLE DOC

- SEATING PLAN OF LAB TEST

Recap of Exceptions

- Catch-or-Specify Requirement

Normal Flow of Execution

```
... /* before, ouside try-catch block */
try {
  o.m(...); /* may throw SomeException */
  ... / * rest of try-block */
}
catch (SomeException se) {
  ... /* rest of catch-block */
}
... /* after, ouside try-catch block */
```

When the exception does not occur

Abnormal Flow of Execution

```
... /* before, ouside try-catch block */
try {
  o.m(...); /* may throw SomeException */
  ... / * rest of try-block */
}
catch (SomeException se) {
  ... /* rest of catch-block */
}
... /* after, ouside try-catch block */
```

When the exception occurs

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 --; }
  }
}
```

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 */
```

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 */
```

What if decrement is implemented incorrectly?

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 --; }
 }
}
```

correct

```
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

```
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 Correct Implementation

```
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 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 */
```

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 */
```

what if one of these throws VTLE unexpectedly

inappropriate ; once an error is discovered, tester should report and terminate right away.

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

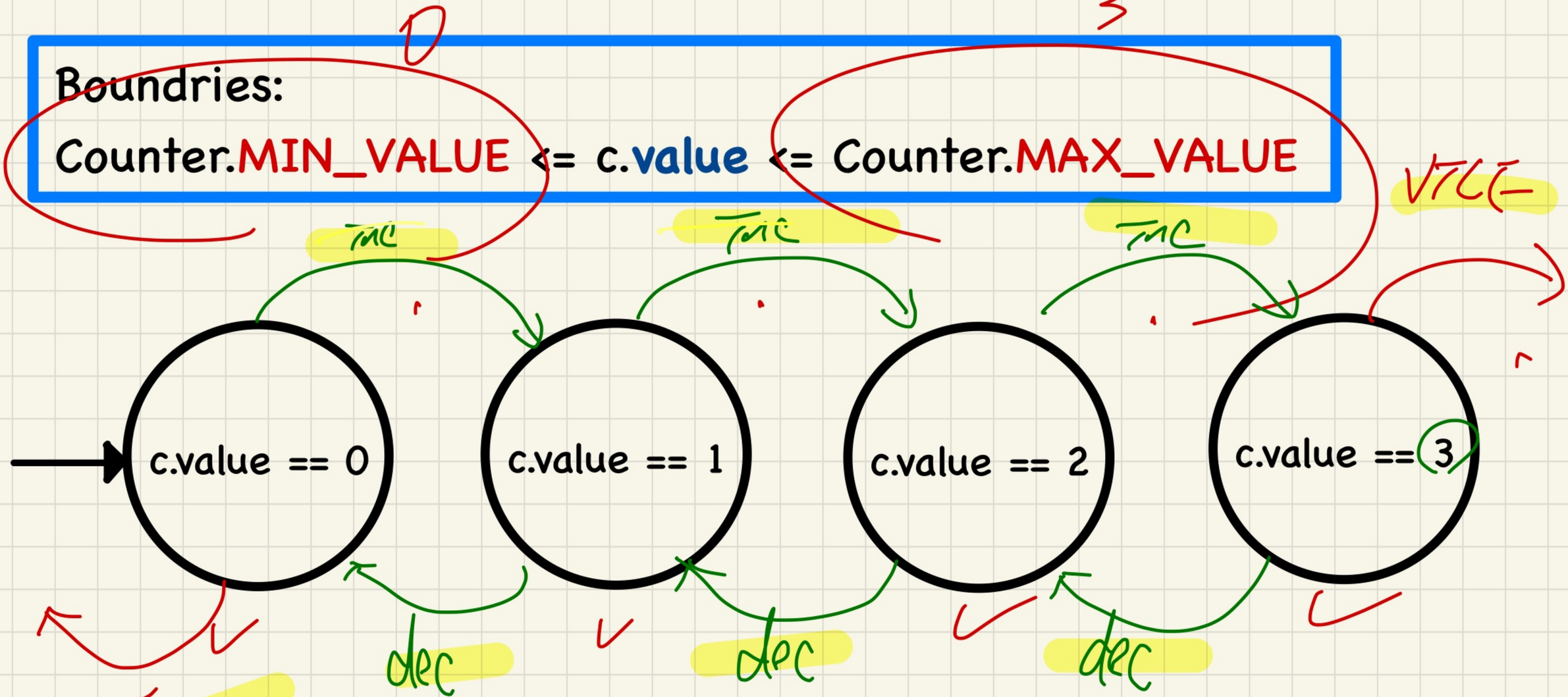
A Manual, Iterative Console Tester

```
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 */
```

Coming Up with Test Cases

Boundries:

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



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

```
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 }
```

What is the easiest way to making this test pass?

JUnit Assertions Examples (2)

Consider the following class:

```
class Circle {
  double radius;
  Circle(double radius) { this.radius = radius; }
  int getArea() { return 3.14 * radius * radius; }
}
```

assertEquals(36.2984, c.getArea()) X

Then consider these assertions. Do they ***pass*** or ***fail***?

```
Circle c = new Circle(3.4);
assertEquals(36.2984, c.getArea(), 0.01);
```

[redacted]

Equals

3.4 * 3.4 * 3.14

36.2984 − 0.01 ≤ c.getArea() ≤ 36.2984 + 0.01

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  }
```

What if method 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 method increment is implemented incorrectly?

JUnit where an Exception is Expected (1)

```
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 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 */
```