

## **Lecture 9 - Oct. 3**

### **TDD with JUnit**

***Parsing Integers***

***Error Handling: Console vs. Exceptions***

***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

## More Example: Parsing Strings as Integers

```
1 Scanner input = new Scanner(System.in);
2 boolean validInteger = false;
3 while (!validInteger) {
4     System.out.println("Enter an integer:");
5     String userInput = input.nextLine();
6     try {
7         int userInteger = Integer.parseInt(userInput);
8         validInteger = true;
9     } catch (NumberFormatException e) {
10        System.out.println(userInput + " is not a valid integer.");
11        /* validInteger remains false */
12    }
13 }
```

Handwritten annotations on the code:

- Line 1: Scanner
- Line 2: boolean validInteger = false;
- Line 3: while (!validInteger) {
- Line 4: System.out.println("Enter an integer:");
- Line 5: String userInput = input.nextLine();
- Line 6: try {
- Line 7: int userInteger = Integer.parseInt(userInput);
- Line 8: validInteger = true;
- Line 9: }
- Line 10: } catch (NumberFormatException e) {
- Line 11: System.out.println(userInput + " is not a valid integer.");
- Line 12: /\* validInteger remains false \*/
- Line 13: }

Additional annotations:

- "twenty-three" written above line 5.
- 23 written below line 5.
- 23 written below line 7.
- "twenty-three" written above line 7.
- NFE (NumberFormatException) written next to line 7.
- 23 written below line 8.

Test Case: ✓

User Enters: twenty-three

User Then Enters: 23 ✓

# Error Handling via Console Messages: Circles

```
1 class Circle {  
2     double radius;  
3     Circle() { /* radius defaults to 0 */ }  
4     void setRadius(double r) {  
5         if (r < 0) { System.out.println("Invalid radius."); }  
6         else { radius = r; }  
7     }  
8     double getArea() { return radius * radius * 3.14; }  
9 }
```

Caller?  
Callee?

① No more CoS Reg. enforced on the callers  
② caller of this method (error originated) is no longer obliged to handle the error.

call stack

```
1 class CircleCalculator {  
2     public static void main(String[] args) {  
3         Circle c = new Circle();  
4         c.setRadius(-10);  
5         double area = c.getArea();  
6         System.out.println("Area: " + area);  
7     }  
8 }
```

C. SR  
CC.m

--- given that the callee already signaled an error, caller still proceeds to calculate

# Error Handling via Console Messages: Banks

```
class Account {
    int id; double balance;
    Account(int id) { this.id = id; /* balance defaults to 0 */ }
    void deposit(double a) {
        if (a < 0) { System.out.println("Invalid deposit."); }
        else { balance += a; }
    }
    void withdraw(double a) {
        if (a < 0 || balance - a < 0) {
            System.out.println("Invalid withdraw."); }
        else { balance -= a; }
    }
}
```

Caller?  
Callee?

call stack

```
class Bank {
    Account[] accounts; int numberOfAccounts;
    Bank(int id) { ... }
    void withdrawFrom(int id, double a) {
        for(int i = 0; i < numberOfAccounts; i++) {
            if(accounts[i].id == id) {
                accounts[i].withdraw(a);
            }
        }
    }
}

class BankApplication {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        Bank b = new Bank(); Account acc1 = new Account(23);
        b.addAccount(acc1);
        double a = input.nextDouble();
        b.withdrawFrom(23, a);
        System.out.println("Transaction Completed.");
    }
}
```

context	caller	callee

## 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 exception*

**Example 2:** Method that calls another which throws the exception

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

## 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) { ... }  
    }  
}
```

**\*\* min a test , and a VICE/VISE happened → pass**  
no VICE/VISE → fail

## Coming Up with Test Cases: A Single, Bounded Variable

**\* min a test , and an exception happened → fail**  
no exception → pass

**\* no exception expected**

Boundries:

Counter.MIN\_VALUE <= c.value <= Counter.MAX\_VALUE

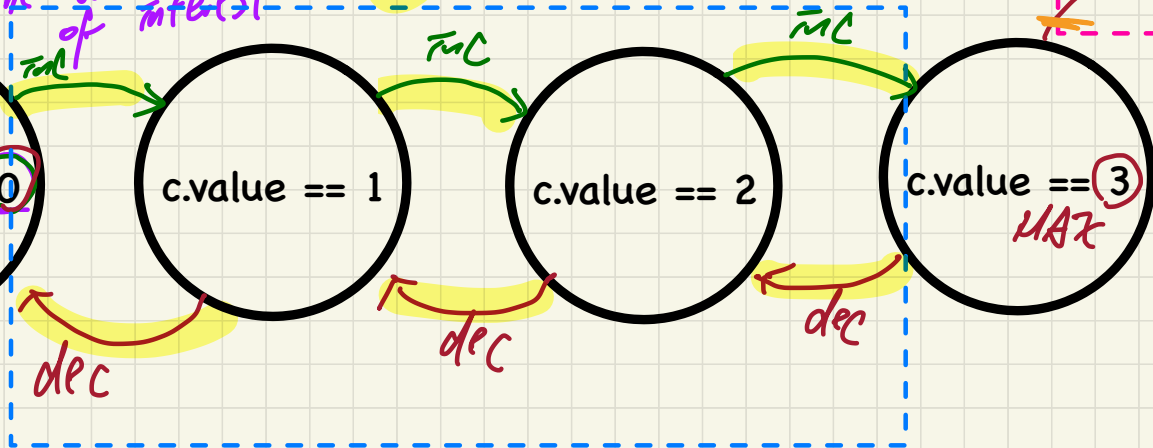
**\*\* exception expected**  
②

TMC

**State diagram** → a combination of the variables of interest

①

init counter obj created.  
dec



→ c.increment()

→ c.decrement()



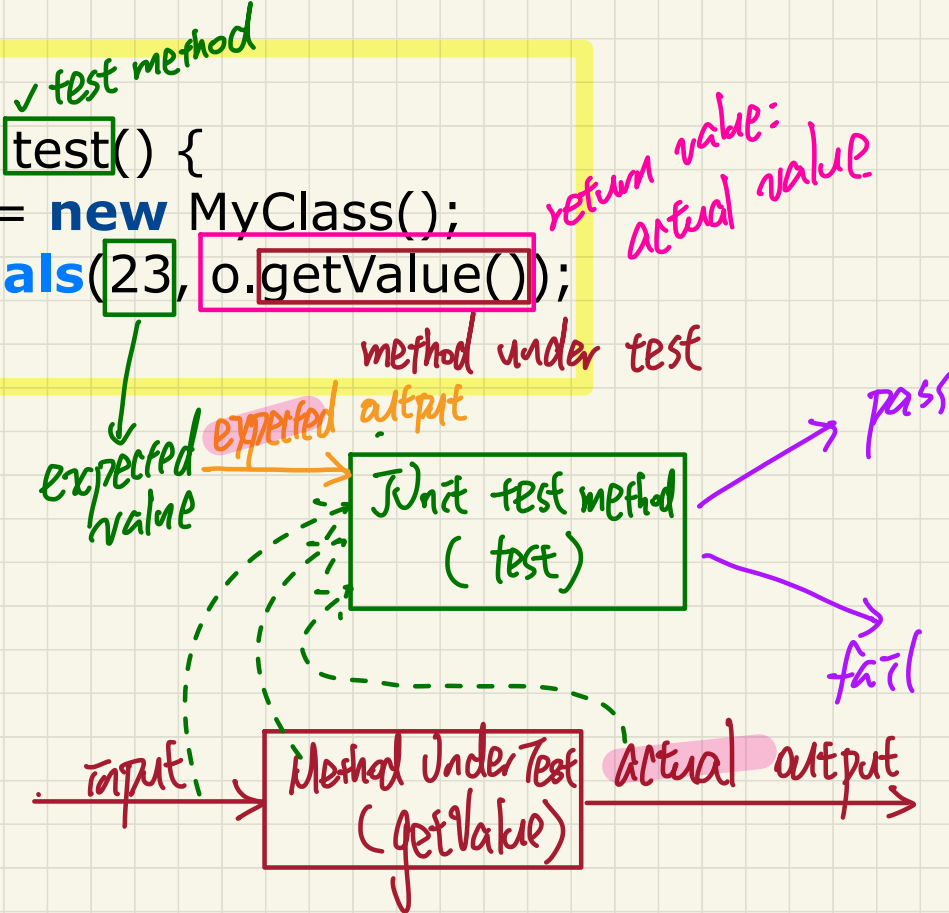
# A Class for Bounded Counters

```
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!
```

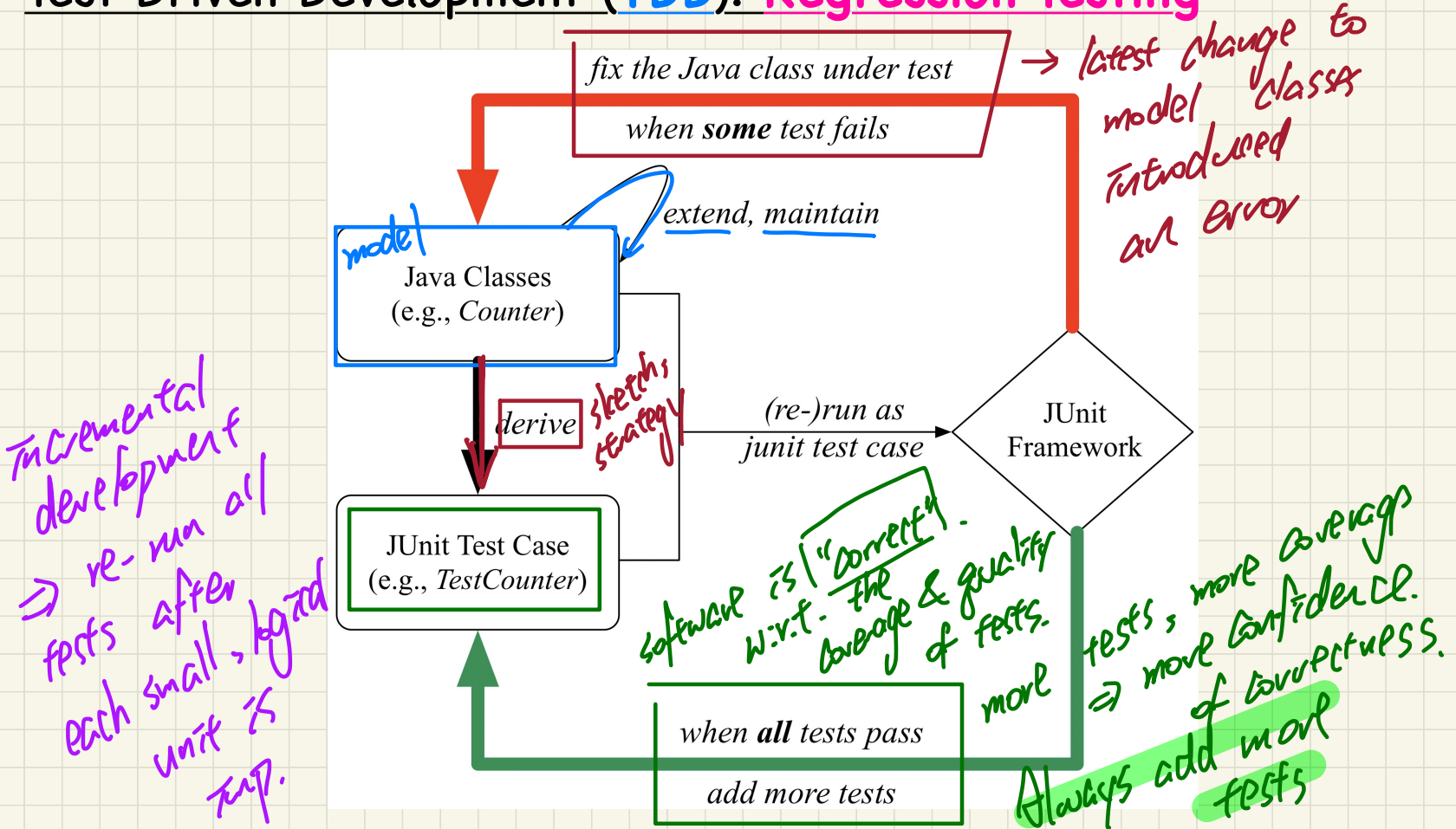
```
/* class Counter */  
    public void increment() throws ValueTooLargeException {  
        if(value == Counter.MAX_VALUE) {  
            throw new ValueTooLargeException("counter value is " + value);  
        }  
        else { value++; }  
    }  
  
    public void decrement() throws ValueTooSmallException {  
        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() {
    MyClass o = new MyClass();
    assertEquals(23, o.getValue());
}
```



# Test-Driven Development (TDD): Regression Testing



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

*Handwritten notes:*  
① not throw VTLE  
② VTLE X  
③ may throw VTLE  
④ fail

What if increment is implemented correctly?

- ① not throw VTLE  
② increment

## 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 }
```

*Handwritten notes:*  
① did not incre.  
② throw VTLE unexpectedly

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 {  
    5 if (value == Counter.MAX_VALUE) {  
        X throw new ValueTooLargeException("counter value is " + value);  
    }  
    6 else { value++; }  
}
```

Handwritten annotations: 0, 0 → 1, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116, 117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129, 130, 131, 132, 133, 134, 135, 136, 137, 138, 139, 140, 141, 142, 143, 144, 145, 146, 147, 148, 149, 150, 151, 152, 153, 154, 155, 156, 157, 158, 159, 160, 161, 162, 163, 164, 165, 166, 167, 168, 169, 170, 171, 172, 173, 174, 175, 176, 177, 178, 179, 180, 181, 182, 183, 184, 185, 186, 187, 188, 189, 190, 191, 192, 193, 194, 195, 196, 197, 198, 199, 200, 201, 202, 203, 204, 205, 206, 207, 208, 209, 210, 211, 212, 213, 214, 215, 216, 217, 218, 219, 220, 221, 222, 223, 224, 225, 226, 227, 228, 229, 230, 231, 232, 233, 234, 235, 236, 237, 238, 239, 240, 241, 242, 243, 244, 245, 246, 247, 248, 249, 250, 251, 252, 253, 254, 255, 256, 257, 258, 259, 260, 261, 262, 263, 264, 265, 266, 267, 268, 269, 270, 271, 272, 273, 274, 275, 276, 277, 278, 279, 280, 281, 282, 283, 284, 285, 286, 287, 288, 289, 290, 291, 292, 293, 294, 295, 296, 297, 298, 299, 300, 301, 302, 303, 304, 305, 306, 307, 308, 309, 310, 311, 312, 313, 314, 315, 316, 317, 318, 319, 320, 321, 322, 323, 324, 325, 326, 327, 328, 329, 330, 331, 332, 333, 334, 335, 336, 337, 338, 339, 340, 341, 342, 343, 344, 345, 346, 347, 348, 349, 350, 351, 352, 353, 354, 355, 356, 357, 358, 359, 360, 361, 362, 363, 364, 365, 366, 367, 368, 369, 370, 371, 372, 373, 374, 375, 376, 377, 378, 379, 380, 381, 382, 383, 384, 385, 386, 387, 388, 389, 390, 391, 392, 393, 394, 395, 396, 397, 398, 399, 400, 401, 402, 403, 404, 405, 406, 407, 408, 409, 410, 411, 412, 413, 414, 415, 416, 417, 418, 419, 420, 421, 422, 423, 424, 425, 426, 427, 428, 429, 430, 431, 432, 433, 434, 435, 436, 437, 438, 439, 440, 441, 442, 443, 444, 445, 446, 447, 448, 449, 450, 451, 452, 453, 454, 455, 456, 457, 458, 459, 460, 461, 462, 463, 464, 465, 466, 467, 468, 469, 470, 471, 472, 473, 474, 475, 476, 477, 478, 479, 480, 481, 482, 483, 484, 485, 486, 487, 488, 489, 490, 491, 492, 493, 494, 495, 496, 497, 498, 499, 500, 501, 502, 503, 504, 505, 506, 507, 508, 509, 510, 511, 512, 513, 514, 515, 516, 517, 518, 519, 520, 521, 522, 523, 524, 525, 526, 527, 528, 529, 530, 531, 532, 533, 534, 535, 536, 537, 538, 539, 540, 541, 542, 543, 544, 545, 546, 547, 548, 549, 550, 551, 552, 553, 554, 555, 556, 557, 558, 559, 560, 561, 562, 563, 564, 565, 566, 567, 568, 569, 570, 571, 572, 573, 574, 575, 576, 577, 578, 579, 580, 581, 582, 583, 584, 585, 586, 587, 588, 589, 590, 591, 592, 593, 594, 595, 596, 597, 598, 599, 600, 601, 602, 603, 604, 605, 606, 607, 608, 609, 610, 611, 612, 613, 614, 615, 616, 617, 618, 619, 620, 621, 622, 623, 624, 625, 626, 627, 628, 629, 630, 631, 632, 633, 634, 635, 636, 637, 638, 639, 640, 641, 642, 643, 644, 645, 646, 647, 648, 649, 650, 651, 652, 653, 654, 655, 656, 657, 658, 659, 660, 661, 662, 663, 664, 665, 666, 667, 668, 669, 670, 671, 672, 673, 674, 675, 676, 677, 678, 679, 680, 681, 682, 683, 684, 685, 686, 687, 688, 689, 690, 691, 692, 693, 694, 695, 696, 697, 698, 699, 700, 701, 702, 703, 704, 705, 706, 707, 708, 709, 710, 711, 712, 713, 714, 715, 716, 717, 718, 719, 720, 721, 722, 723, 724, 725, 726, 727, 728, 729, 730, 731, 732, 733, 734, 735, 736, 737, 738, 739, 740, 741, 742, 743, 744, 745, 746, 747, 748, 749, 750, 751, 752, 753, 754, 755, 756, 757, 758, 759, 760, 761, 762, 763, 764, 765, 766, 767, 768, 769, 770, 771, 772, 773, 774, 775, 776, 777, 778, 779, 780, 781, 782, 783, 784, 785, 786, 787, 788, 789, 790, 791, 792, 793, 794, 795, 796, 797, 798, 799, 800, 801, 802, 803, 804, 805, 806, 807, 808, 809, 810, 811, 812, 813, 814, 815, 816, 817, 818, 819, 820, 821, 822, 823, 824, 825, 826, 827, 828, 829, 830, 831, 832, 833, 834, 835, 836, 837, 838, 839, 840, 841, 842, 843, 844, 845, 846, 847, 848, 849, 850, 851, 852, 853, 854, 855, 856, 857, 858, 859, 860, 861, 862, 863, 864, 865, 866, 867, 868, 869, 870, 871, 872, 873, 874, 875, 876, 877, 878, 879, 880, 881, 882, 883, 884, 885, 886, 887, 888, 889, 890, 891, 892, 893, 894, 895, 896, 897, 898, 899, 900, 901, 902, 903, 904, 905, 906, 907, 908, 909, 910, 911, 912, 913, 914, 915, 916, 917, 918, 919, 920, 921, 922, 923, 924, 925, 926, 927, 928, 929, 930, 931, 932, 933, 934, 935, 936, 937, 938, 939, 940, 941, 942, 943, 944, 945, 946, 947, 948, 949, 950, 951, 952, 953, 954, 955, 956, 957, 958, 959, 960, 961, 962, 963, 964, 965, 966, 967, 968, 969, 970, 971, 972, 973, 974, 975, 976, 977, 978, 979, 980, 981, 982, 983, 984, 985, 986, 987, 988, 989, 990, 991, 992, 993, 994, 995, 996, 997, 998, 999, 1000

```
1  @Test  
2  public void testIncAfterCreation() {  
3      1 Counter c = new Counter();  
4      2 assertEquals(Counter.MIN_VALUE, c.getValue());  
5      3 try { c.v == 0 ✓  
6          4 c.increment(); I  
7          5 assertEquals(1, c.getValue());  
8      }  
9      X catch (ValueTooLargeException e) {  
10         /* Exception is not expected to be thrown. */  
11         X fail ("ValueTooLargeException is not expected.");  
12     }  
13     6
```

# Running JUnit Test 1 on Incorrect Implementation

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

```
1  @Test  
2  public void testIncAfterCreation() {  
3      1 Counter c = new Counter();  
4      2 assertEquals(Counter.MIN_VALUE, c.getValue());  
5      3 try { c.v == 0  
6          4 c.increment();  
7          X assertEquals(1, c.getValue());  
8      }  
9      7 catch (ValueTooLargeException e) {  
10         /* Exception is not expected to be thrown. */  
11         8 fail("ValueTooLargeException is not expected.");  
12     }  
13 }
```

↳ reject the unexpected VILE by failing the test