Lecture 11 - June 10

TDD with JUnit, *Object Equality*

Console Errors vs. fail Assertions
Two vs. Single try-catch Blocks
Patternizing Assertions using Loops
Call by Value

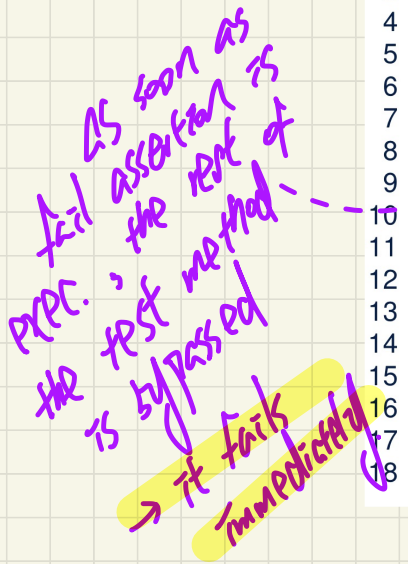
Announcements/Reminders

- Today's class: notes template posted
- ProgTest0 marks & feedback released
- ProgTest1 this Friday **JUN 13**
- ProgTest1 guide (policies & requirements) released
- PracticeTest1 released
- **In-Person** Review session:
  + 4 PM @ CLH M, Wednesday, June 11
- Priorities:
  + Lab1 solution, Lab2
  + Slides on **Classes and Objects**
  + Slides on **Exceptions**

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 */
```

what if: a VTLE thrown unexpectedly.

Assume. increment 4th call throws VTLE

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

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 }
```

p1: VTLE not expected

p2: VTLE expected

pass or fail?

VTLE from *p1 → fail

VTLE from **p2 → pass

→ try-catch blocks * 2 needed.

Hint: Say Line 12 is executed,

is it clear if that ValueTooLargeException was thrown as expected?

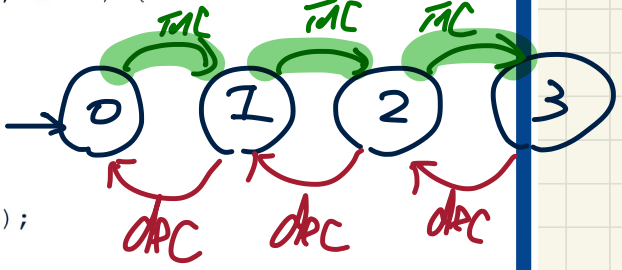
Testing Many Values in a Single Test

useful when the range between MIN and MAX is large.

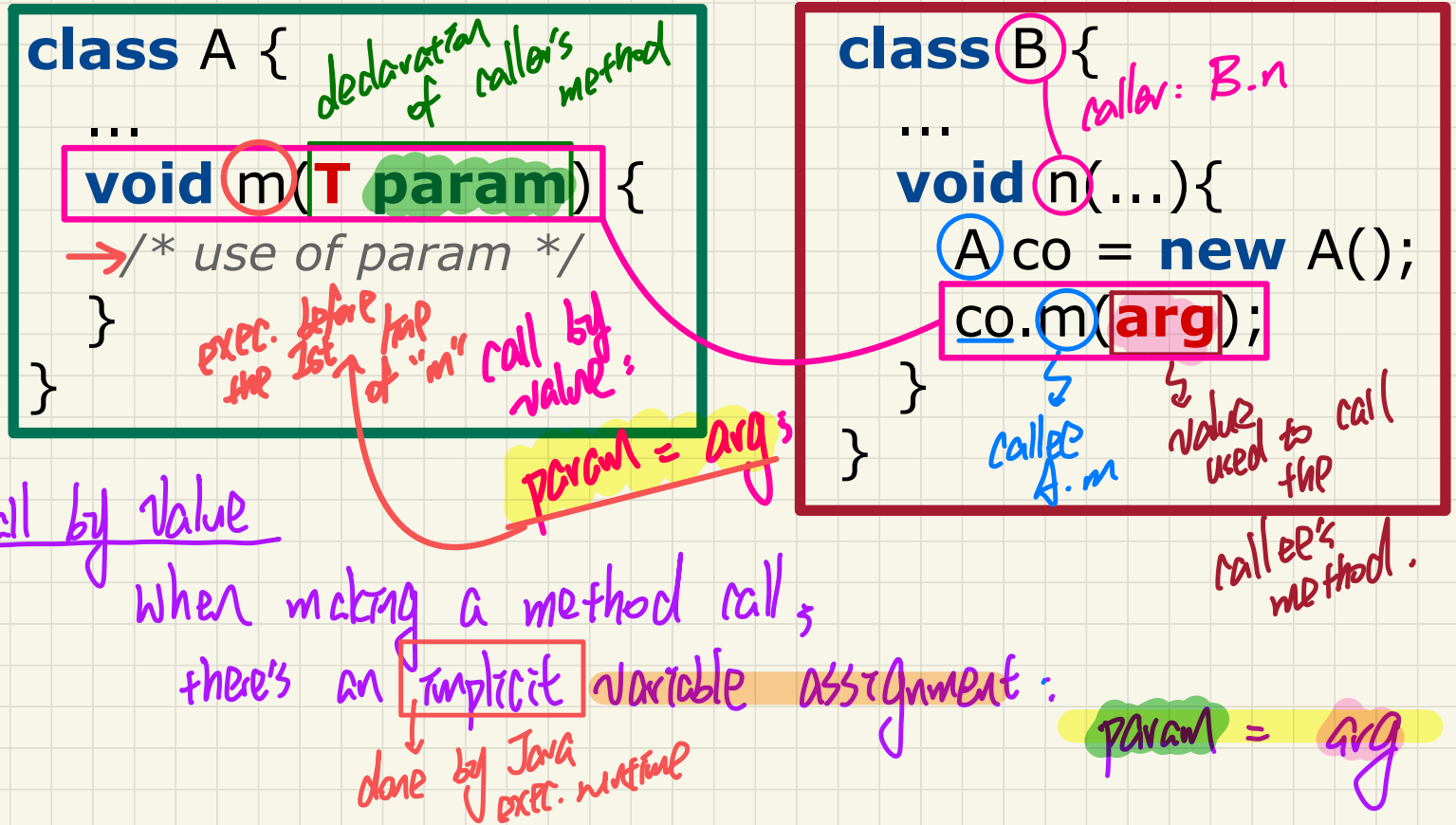
Loops can make it effective on generating test cases:

```
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 }
```

i: 0, 1, 2



Method Call: Callee vs. Caller

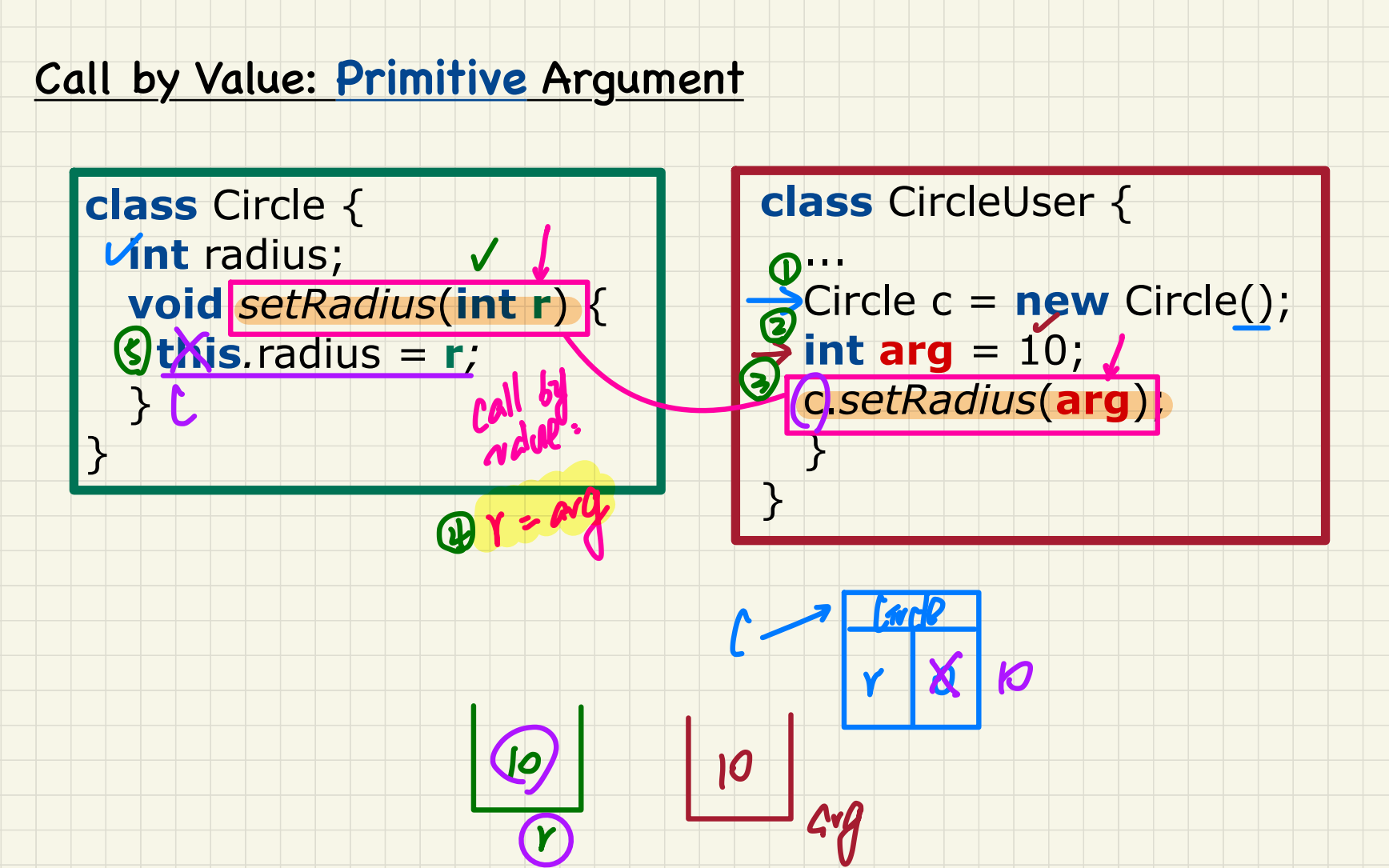


Call by Value: Primitive Argument

```
class Circle {
  int radius;
    void setRadius(int r) {
      this.radius = r;
    }
}
```

```
class CircleUser {
  ...
  Circle c = new Circle();
  int arg = 10;
  c.setRadius(arg);
  }
}
```

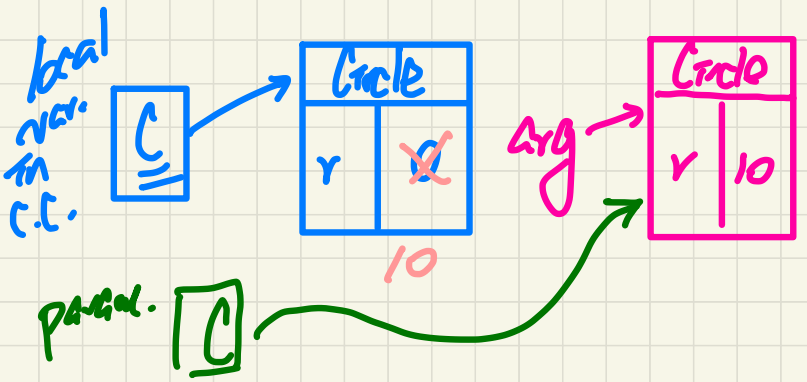
call by value.



Call by Value: Reference Argument

```
class Circle {
   int radius;
   Circle() {}
   Circle(int r) {
      this.radius = r;
   }
   void setRadius(Circle c) {
      this.radius = c.radius;
   }
}
```

```
class CircleUser {
   ...
   Circle c = new Circle();
   Circle arg = new Circle(10);
   c.setRadius(arg);
   }
}
```



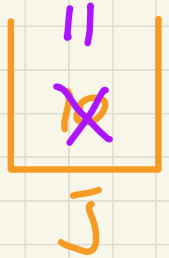
Call by Value: Re-Assigning Primitive Parameter

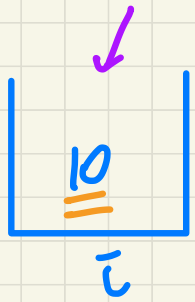
```
public class Util {
  void reassignInt(int j) {
    j = j + 1; }
  void reassignRef(Point q) {
    Point np = new Point(6, 8);
    q = np; }
  void changeViaRef(Point q) {
    q.moveHorizontally(3);
    q.moveVertically(4); } }
```

```
1  @Test
2  public void testCallByVal() {
3    Util u = new Util();
4    int i = 10;
5    assertTrue(i == 10);
6    u.reassignInt(i);
7    assertTrue(i == 10);
8  }
```

C.B.V.: J=i

not modifying the argument value, instead, modifying a copy of it.





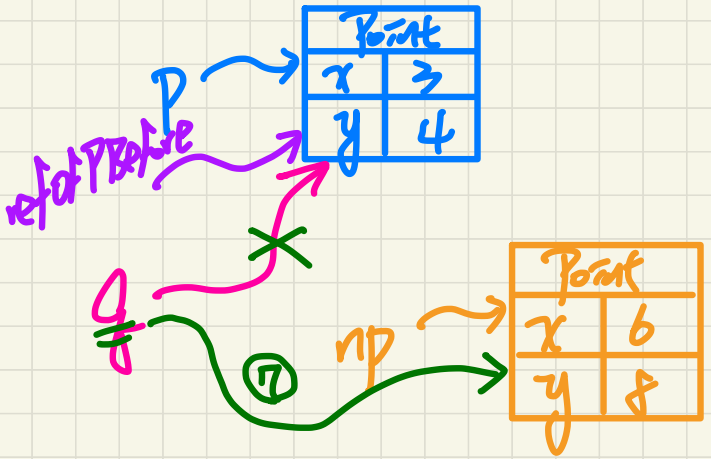
Call by Value: Re-Assigning Reference Parameter

```
public class Util {
  void reassignInt(int j) {
    j = j + 1; }
  void reassignRef(Point q) {
    Point np = new Point(6, 8);
    q = np; }
  void changeViaRef(Point q) {
    q.moveHorizontally(3);
    q.moveVertically(4); } }
```

C.B.V. q=p

```
1  @Test
2  public void testCallByRef_1() {
3    Util u = new Util();
4    Point p = new Point(3, 4);
5    Point refOfPBefore = p;
6    u.reassignRef(p);
7    assertTrue(p == refOfPBefore);
8    assertTrue(p.getX() == 3);
9    assertTrue(p.getY() == 4);
10 }
```

not reassigned; only a copy of it (q) re-assigned

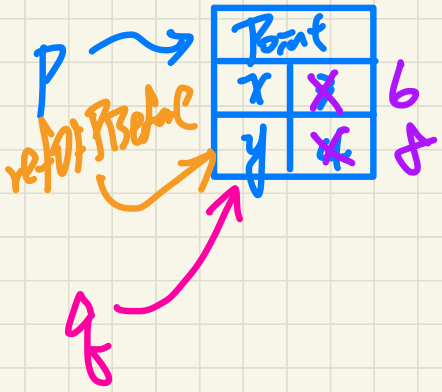



```
public class Point {
  private int x;
  private int y;
  public Point(int x, int y) {
    this.x = x;
    this.y = y;
  }
  public int getX() { return this.x; }
  public int getY() { return this.y; }
  public void moveVertically(int y){ this.y += y; }
  public void moveHorizontally(int x){ this.x += x; }
}
```

Call by Value: Calling Mutator on Reference Parameter

```
public class Util {
  void reassignInt(int j) {
    j = j + 1; }
  void reassignRef(Point q) {
    Point np = new Point(6, 8);
    q = np; }
  void changeViaRef(Point q) {
    q.moveHorizontally(3);
    q.moveVertically(4); } }
```

```
1  @Test
2  public void testCallByRef_2() {
3    Util u = new Util();
4    Point p = new Point(3, 4);
5    Point refOfPBefore = p;
6    u.changeViaRef(p);
7    assertTrue(p == refOfPBefore);
8    assertTrue(p.getX() == 6);
9    assertTrue(p.getY() == 8);
10 }
```



C.B.V.

```
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; }
  public void moveVertically(int y){ this.y += y; }
  public void moveHorizontally(int x){ this.x += x; }
}
```