Lecture 14 - June 26

Object Equality

Short Circuit Evaluation in equals
JUnit: assertSame vs. assertEquals

Announcements/Reminders

- Today's class: notes template posted
- On-demand extra TA help hours
- WrittenTest1 tomorrow
  + Review session materials released
- Priorities:
  + Lab2 solution, Lab3

Short-Circuit Evaluation: ||

Left Operand op1	Right Operand op2	op1 || op2
false	false	false
true	false	true
false	true	true
true	true	true

T → avoid/skip eval: 10/0

Test Inputs:

→ x = 0, y = 10

x = 5, y = 10

```
System.out.println("Enter x:");
int x = input.nextInt();
System.out.println("Enter y:");
int y = input.nextInt();
if(x == 0 || y / x > 2) {
    if(x == 0) {
        System.out.println("Error: Division by Zero");
    }
    else {
        System.out.println("y / x is greater than 2");
    }
}
else { /* !(x == 0 || y / x > 2) == (x != 0 && y / x <= 2) */
    System.out.println("y / x is not greater than 2");
}
```

0 10 0

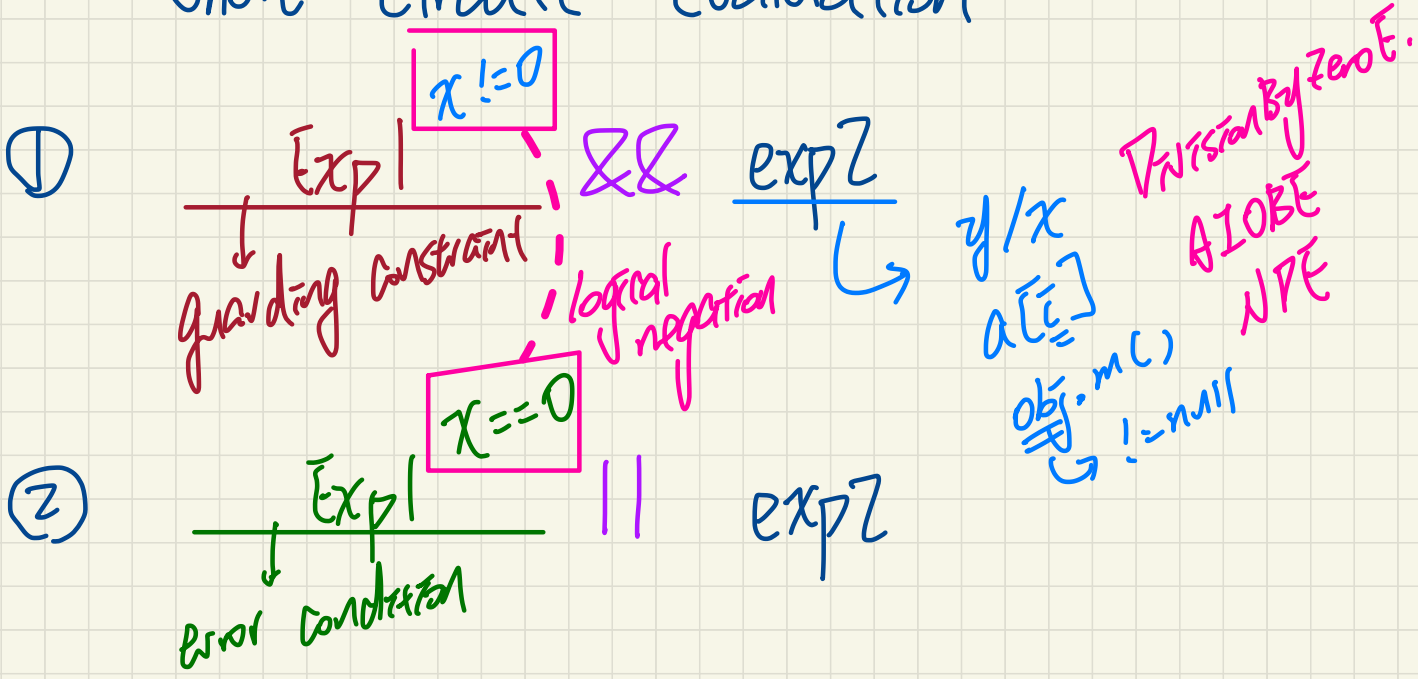
eval: exp1 || exp2

① Eval exp1 → T

(2.1) Eval exp1 → T
↳ skip eval exp2
↳ || → T

(2.2) Eval exp1 → F
↳ eval exp2 to decide.

Short Circuit Evaluation



Short-Circuit Evaluation: Common Errors

Test Inputs:
x = 0, y = 10

Short-Circuit Evaluation is not exploited: crash when `x == 0`

```
if (y / x > 2 && x != 0) {
  /* do something */
}
else {
  /* print error */ }
```

Eval: L→R

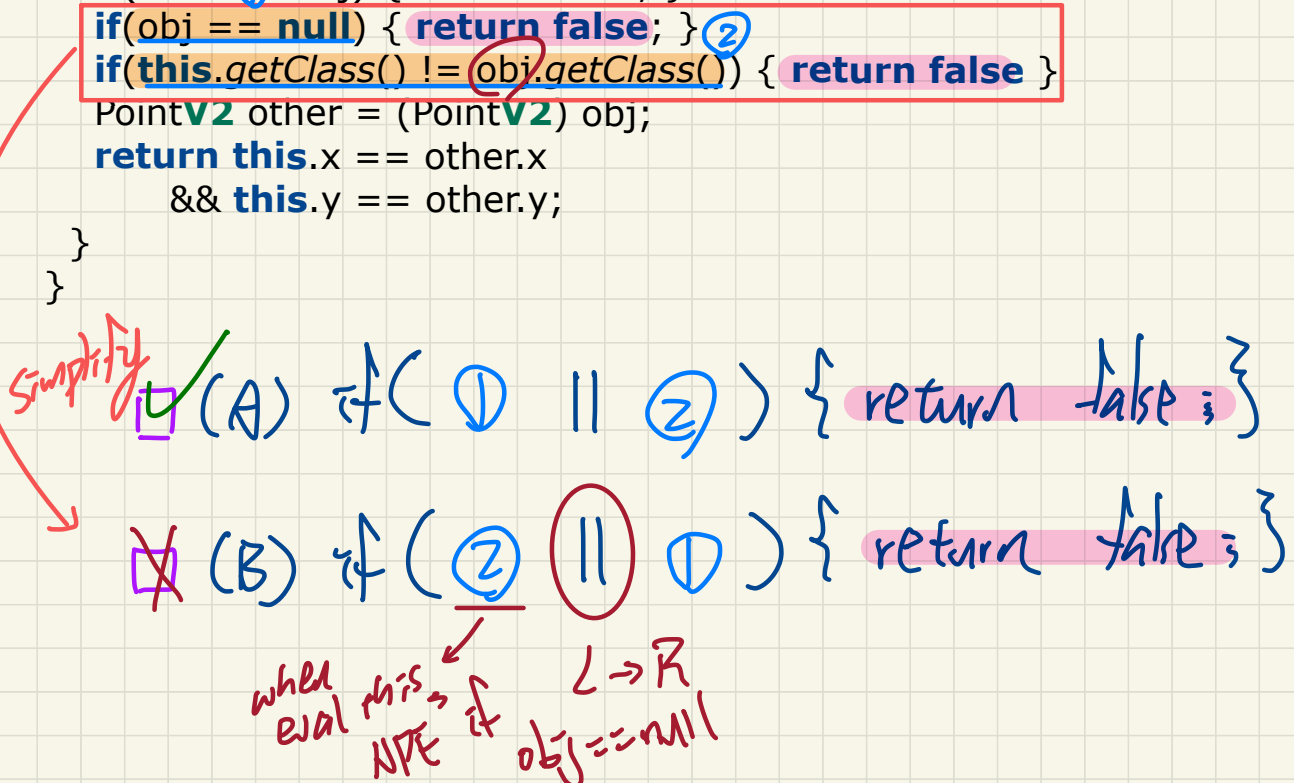
10/0 > 2

↳ DBZE!

Short-Circuit Evaluation is not exploited: crash when `x == 0`

```
if (y / x <= 2 || x == 0) {
  /* print error */
}
else {
  /* do something */ }
```

```java
public class PointV2 {
  private int x; double y;
  public PointV2 (double x, double y) { ... }
  boolean equals(Object obj) {
    if(this == obj) { return true; }
    if(obj == null) { return false; }
    if(this.getClass() != obj.getClass()) { return false }
    PointV2 other = (PointV2) obj;
    return this.x == other.x
        && this.y == other.y;
  }
}
```



The equals Method:

To Override or Not?

```
public class Object {
   ...
   public boolean equals(Object obj) {
      return this == obj;
   }
}
```

extends

```
public class PointV1 {
   private int x;
   private int y;
   public PointV1 (int x, int y) {
      this.x = x;
      this.y = y;
   }
}
```

extends

```
public class PointV2 {
   private int x; double y;
   public PointV2 (double x, double y) { ... }
   boolean equals(Object obj) {
      if(this == obj) { return true; }
      if(obj == null) { return false; }
      if(this.getClass() != obj.getClass()) { return false }
      PointV2 other = (PointV2) obj;
      return this.x == other.x
           && this.y == other.y;
   }
}
```

* DT of p1 is PointV1 → default equals from Object is invoked

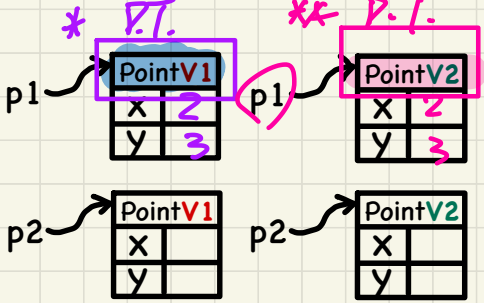
** DT of p1 is PointV2 → overridden ver. is invoked

```
1  String s = "(2, 3)";
2  PointV1 p1 = new PointV1(2, 3);
3  PointV1 p2 = new PointV1(2, 3);
4  PointV1 p3 = new PointV1(4, 6);
5  System.out.println(p1 == p2);   /* false */
6  System.out.println(p2 == p3);   /* false */
7  System.out.println(p1.equals(p1));   /* true */
8  System.out.println(p1.equals(null));   /* false */
9  System.out.println(p1.equals(s));   /* false */
10 System.out.println(p1.equals(p2));   /* false */
11 System.out.println(p2.equals(p3));   /* false */
```

D.T.?

```
1  String s = "(2, 3)";
2  PointV2 p1 = new PointV2(2, 3);
3  PointV2 p2 = new PointV2(2, 3);
4  PointV2 p3 = new PointV2(4, 6);
5  System.out.println(p1 == p2);   /* false */
6  System.out.println(p2 == p3);   /* false */
7  System.out.println(p1.equals(p1));   /* true */
8  System.out.println(p1.equals(null));   /* false */
9  System.out.println(p1.equals(s));   /* false */
10 System.out.println(p1.equals(p2));   /* true */
11 System.out.println(p2.equals(p3));   /* false */
```

D.T.?



** not always true :

The equals Method: Overridden Version

P⇒Q (logical implication)

P→Q

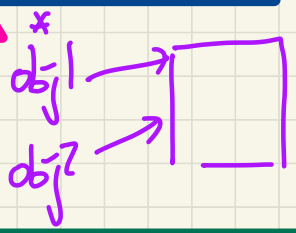
Example 2

```
public class Object {
  ...
  public boolean equals(Object obj) {
    return this == obj;
  }
}
```

implication

T⇒F ≡ F

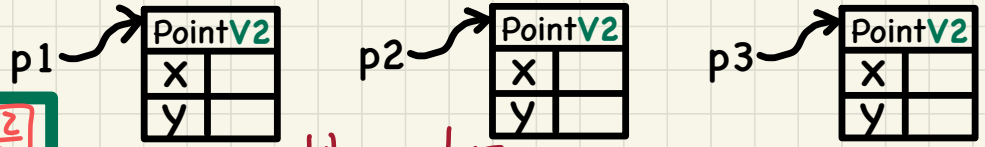
extends



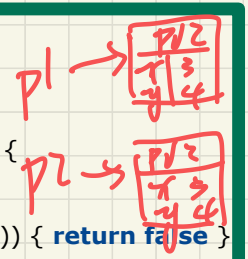

```
1  PointV2 p1 = new PointV2(3, 4);
2  PointV2 p2 = new PointV2(3, 4);
3  PointV2 p3 = new PointV2(4, 5);
4  System.out.println(p1 == p1);   /* [redacted] */
5  System.out.println(p1.equals(p1));   /* [redacted] */
6  System.out.println(p1 == p2);   /* [redacted] */
7  System.out.println(p1.equals(p2));   /* [redacted] */
8  System.out.println(p2 == p3);   /* [redacted] */
9  System.out.println(p2.equals(p3));   /* [redacted] */
```

(A) is F

(B) is T




```
public class PointV2 {
  private int x;
  private int y;
  public PointV2 (int x, int y) { ... }
  public boolean equals(Object obj) {
    if(this == obj) { return true; }
    if(obj == null) { return false; }
    if(this.getClass() != obj.getClass()) { return false }
    PointV2 other = (PointV2) obj;
    return this.x == other.x
        && this.y == other.y;
  }
}
```



obj1 == obj2

(A) Two objects are reference-equal.

(B) Two objects are contents-equal.

obj1.eq(obj2)

A ⇒ B

If (A) is true, then (B) is true. ✓

If (B) is true, then (A) is true. ✗ B ⇒ A

T

x.equals(null) → return false
(phase 2)

x != null

↳ ∴ x cannot be null

↳ otherwise: NPE!

(1) if (x != null) { x.equals(null); }

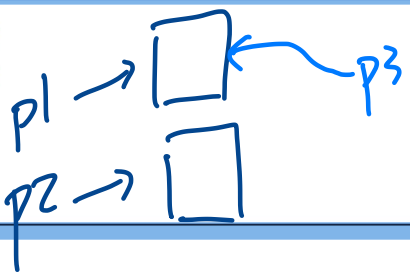
(2) x == null || x.equals(null)

assertSame vs. assertEquals

assertSame(exp1, exp2) → exp1 == exp2 → T → pass; → F → fail

- Passes if exp1 and exp2 are references to the same object
  - ≈ **assertTrue**(exp1 == exp2)
  - ≈ **assertFalse**(exp1 != exp2) → exercise: why does this work?

```
PointV1 p1 = new PointV1(3, 4);
PointV1 p2 = new PointV1(3, 4);
PointV1 p3 = p1;
assertSame(p1, p3);   pass
assertSame(p2, p3);   fail
```



assertEquals(exp1, exp2)

- ≈ exp1 == exp2 if exp1 and exp2 are **primitive** type

```
int i = 10;
int j = 20;
assertEquals(i, j);   fail
```

comp. values

ref. types

assertEquals(exp1 , exp2)

↳ exp1.equals(exp2)

↳ version invoked depends on exp1's DT

① assertEquals(exp1 , exp2)

↳ exp1 (DT) equals(exp2) → equals from exp1's DT called

② assertEquals(exp2 , exp1)

↳ exp2 (DT) equals(exp1) → equals from exp2's DT called.

assertEquals: Reference Comparison or Not

assertEquals(exp1, exp2)

- ≈ `exp1.equals(exp2)` if exp1 and exp2 are ***reference*** type

Case 1: If `equals` is ***not*** explicitly overridden in `exp1`'s declared type

≈ ***assertSame***(exp1, exp2)

```
PointV1 p1 = new PointV1(3, 4);
PointV1 p2 = new PointV1(3, 4);
PointV2 p3 = new PointV2(3, 4);
assertEquals(p1, p2);
assertEquals(p2, p3);
```

Case 2: If `equals` is explicitly ***overridden*** in `exp1`'s declared type

≈ `exp1.equals(exp2)`

```
PointV1 p1 = new PointV1(3, 4);
PointV1 p2 = new PointV1(3, 4);
PointV2 p3 = new PointV2(3, 4);
assertEquals(p1, p2);
assertEquals(p2, p3);
assertEquals(p3, p2);
```

* Object version

↳ p1 == p2

↳ false

** Object version

↳ p2 == p1

* p2.equals(p3)

↓

D.T. PV1 → object version.

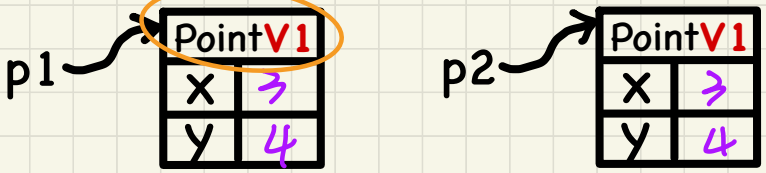
** p3.equals(p2)

D.T. PV2 → overridden equals from PV2 is called.

Testing Default Equality of Points in JUnit

```
@Test
public void testEqualityOfPointV1() {
  PointV1 p1 = new PointV1(3, 4); PointV1 p2 = new PointV1(3, 4);
  assertFalse(p1 == p2); assertFalse(p2 == p1);
  /* assertSame(p1, p2); assertSame(p2, p1); */ /* both fail */
  assertFalse(p1.equals(p2)); assertFalse(p2.equals(p1));
  assertTrue(p1.getX() == p2.getX() && p2.getY() == p2.getY());
}
```

pass
fail
F
Object version
p1 == p2
F




```
public class Object {
  ...
  public boolean equals(Object obj) {
    return this == obj;
  }
}
```

extends

```
public class PointV1 {
  private int x;
  private int y;
  public PointV1 (int x, int y) {
    this.x = x;
    this.y = y;
  }
}
```

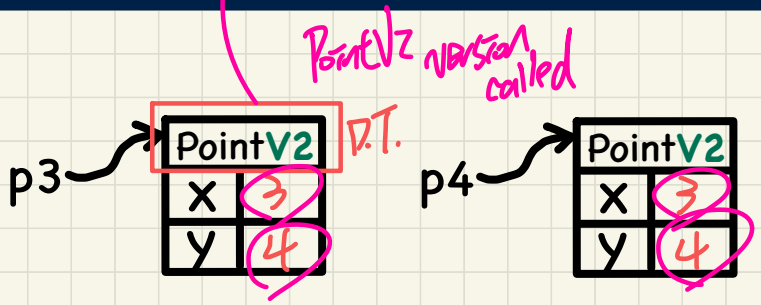
Testing Overridden Equality of Points in JUnit

```
@Test
public void testEqualityOfPointV2() {
  PointV2 p3 = new PointV2(3, 4); PointV2 p4 = new PointV2(3, 4);
  assertFalse(p3 == p4); assertFalse(p4 == p3);
  /* assertSame(p3, p4); assertSame(p4, p4); */ /* both fail */
  assertTrue(p3.equals(p4)); assertTrue(p4.equals(p3));
  assertEquals(p3, p4); assertEquals(p4, p3);
}
```

```
public class Object {
  ...
  public boolean equals(Object obj) {
    return this == obj;
  }
}
```

extends

```
public class PointV2 {
  private int x;
  private int y;
  public PointV2 (int x, int y) { ... }
  public boolean equals(Object obj) {
    if(this == obj) { return true; }
    if(obj == null) { return false; }
    if(this.getClass() != obj.getClass()) { return false }
    Point other = (PointV2) obj;
    return this.x == other.x
        && this.y == other.y;
  }
}
```

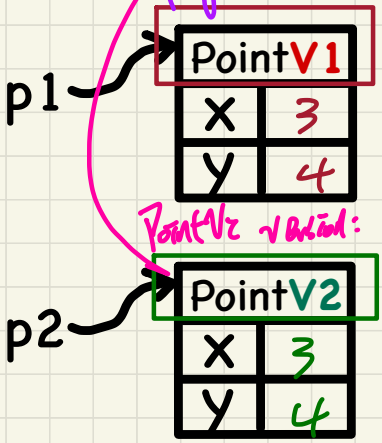


Testing Equality of Points in JUnit: Default vs. Overridden

```java
@Test
public void testEqualityOfPointV1andPointv2() {
  PointV1 p1 = new PointV1(3, 4); PointV2 p2 = new PointV2(3, 4);
  /* These two assertions do not compile because p1 and p2 are of different types. */
  /* assertFalse(p1 == p2); assertFalse(p2 == p1); */
  /* assertSame can take objects of different types and fail. */
  /* assertSame(p1, p2); */ /* compiles, but fails */
  /* assertSame(p2, p1); */ /* compiles, but fails */
  /* version of equals from Object is called */
  assertFalse(p1.equals(p2));
  /* version of equals from PointP2 is called */
  assertFalse(p2.equals(p1));
}
```

```java
public class Object {
  ...
  public boolean equals(Object obj) {
    return this == obj;
  }
}
```

Object Version: → p1 == p2" → F



extends extends

```java
public class PointV1 {
  private double x;
  private double y;
  public PointV1 (double x, double y) {
    this.x = x;
    this.y = y;
  }
}
```

```java
public class PointV2 {
  private int x; private int y;
  public PointV2 (int x, int y) { ... }
  public boolean equals(Object obj) {
    if(this == obj) { return true; }
    if(obj == null) { return false; }
    if(this.getClass() != obj.getClass()) { return false }
    PointV2 other = (PointV2) obj;
    return this.x == other.x
        && this.y == other.y;
  }
}
```