

## Lecture 13 - June 24

### Object Equality

**Static Type, Dynamic Type, Type Casting  
equals: Overridden Version Phases 3, 4  
Short-Circuit Evaluation**

## Announcements/Reminders

- Today's class: notes template posted
- On-demand extra TA help hours
- **WrittenTest1** this Friday
  - + In-Person Review Session: 1 PM, WED, June 25 (CLH H)
  - + Googledoc to post your questions (see lectures site)
- Priorities:
  - + Lab2 solution, Lab3

may be changed  $\longleftrightarrow$  multiple type: determines the version of method called.

## Static Type, Dynamic Type, Type Casting

fixed  
↑  
declared type

↓  
determines  
the range  
of methods  
that can  
be called  
on a variable

**Static Type** a ref variable's declared type

**Dynamic Type** type of address currently stored in a ref variable

**Type Casting**: creating an expression of certain **static type**

```
class C1 {
    ...
    public void m1() {...}
}

class C2 {
    ...
    public void m2() {...}
}
```

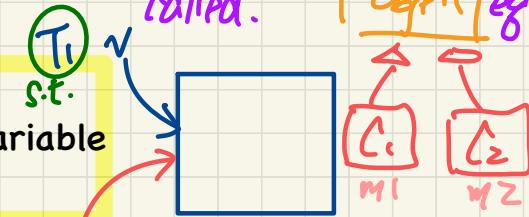
s.t.  $C_1$  o1 = new  $C_1()$ ;  
 $C_2$  o2 = new  $C_2()$ ;

o1.m1(); ✓ ∵ m1 ded. in o1's ST  
o1.m2(); X m1 ded. in o1's ST  
o2.m1(); X m2 undefined in o2's ST  
o2.m2(); ✓ in o2's ST

**Object** o3;

\* o3 = o1;  $\rightarrow$  o3's ST remains Object  
o3.m1(); X  
o3.m2(); X

((C1) o3).m1(); ✓ ∵ the type cast ((C1) o3) has ST: C1  
((C1) o3).m2(); X  
o3 = o2;  
((C2) o3).m1(); X  
((C2) o3).m2(); ✓ ∵ the type cast ((C2) o3) has ST: C2



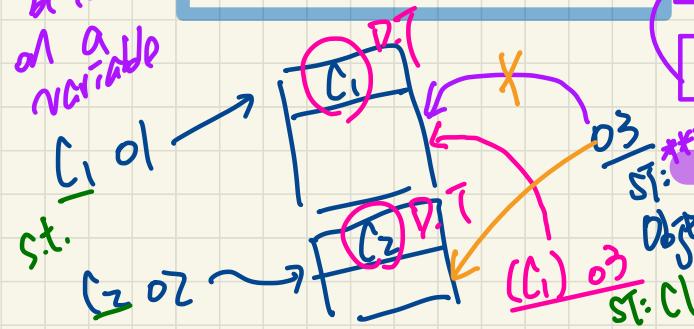
(T2) ✓  
type cast f v  
⇒ alias to the  
same object but  
with a diff. ST.

\* o3 has ST: Object.  
↳ o3.equals ✓

↳ o3.m1 X  
↳ o3.m2 X

cast ((C1) o3)  
has ST: C1

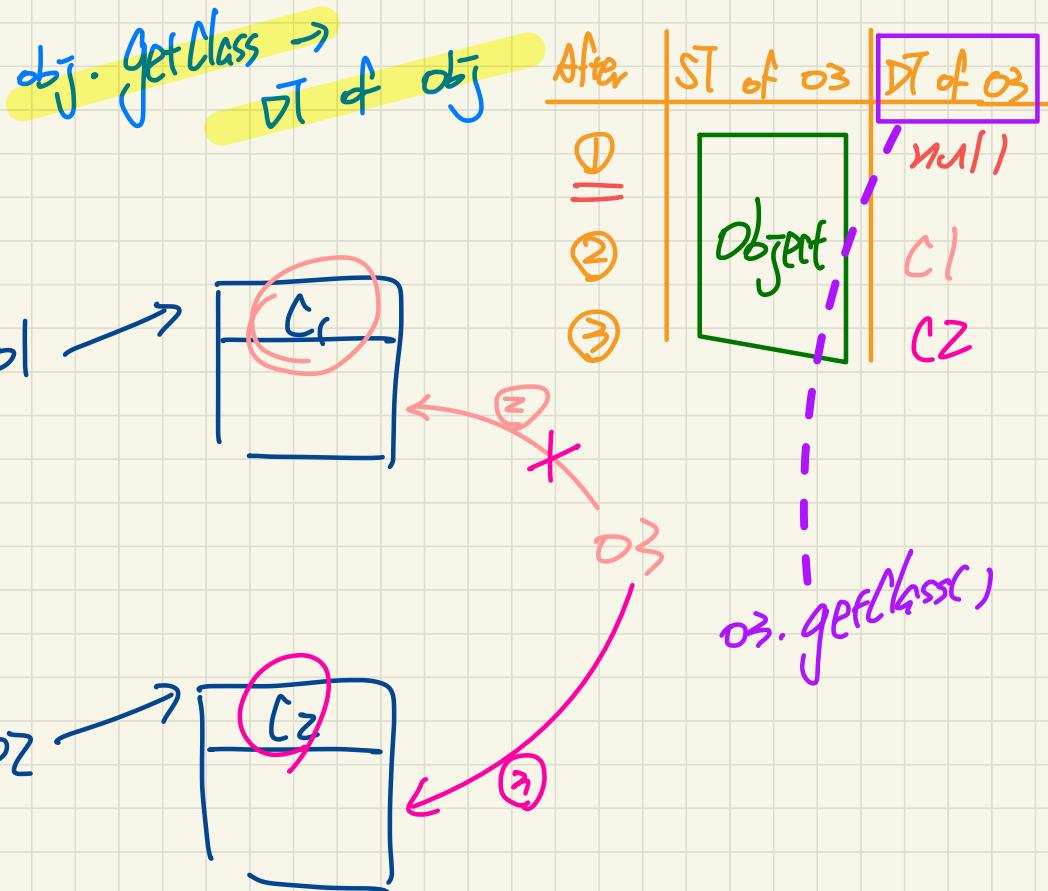
After \*: DT of o3 is C1  
After \*\*: DT of o3 is C2



```

C1 o1 = new C1();
C2 o2 = new C2();
o1.m1();
o1.m2();
o2.m1();
o2.m2();
① Object o3;
② o3 = o1;
③ o3.m1();
④ o3.m2();
⑤ o3 = o2;
⑥ ((C2) o3).m1();
⑦ ((C2) o3).m2();

```



# The `equals` Method: Overridden Version

Phase 3

```
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 }  
        PointV2 other = (PointV2) obj;  
        return this.x == other.x  
            && this.y == other.y;  
    }  
}
```

① `this != obj`  
② `obj != null`

`this.getClass()`

`obj.getClass()`

`this`

`obj`

`objects with`

`different DTs are not equal to each other`

PointV2
X
Y

# The `equals` Method: Overridden Version

Example 1: Trace L9

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

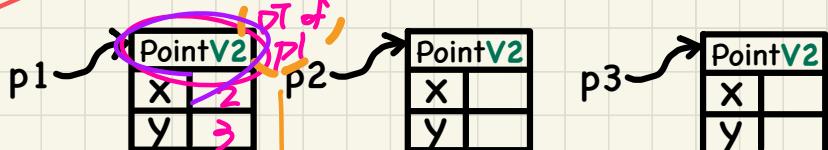
extends

```
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 }  
        PointV2 other = (PointV2) obj;  
        return this.x == other.x  
            && this.y == other.y;  
    }  
}
```

Diagram showing the call by value mechanism:

- A pink arrow points from the `Object.equals` method up to the `PointV2.equals` method.
- A red arrow labeled "call by value" points from the `obj` parameter in the `Object.equals` method down to the `obj` parameter in the `PointV2.equals` method.
- Handwritten annotations:
  - "P1" is written next to the `p1` variable.
  - "PointV2" is written next to the `PointV2` class name.
  - "S" is written next to the string literal `"(2, 3)"`.
  - "obj = S" is written next to the assignment statement `obj = s;`
  - "returning" is written next to the `return` statement in the `PointV2.equals` method.

```
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); /* [REDACTED] */  
6 System.out.println(p2 == p3); /* [REDACTED] */  
7 System.out.println(p1.equals(p1)); /* [REDACTED] */  
8 System.out.println(p1.equals(null)); /* [REDACTED] */  
9 System.out.println(p1.equals(s)); /* [REDACTED] */  
10 System.out.println(p1.equals(p2)); /* [REDACTED] */  
11 System.out.println(p2.equals(p3)); /* [REDACTED] */
```



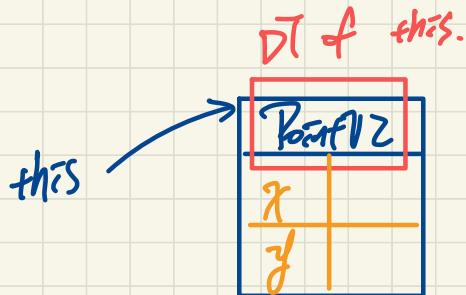
Annotations in orange and yellow:

- "P1 of P1" is written above the `p1` variable.
- "P1 of P2" is written above the `p2` variable.
- "P3" is written above the `p3` variable.
- "obj" is written next to the `obj` parameter in the `PointV2.equals` method.
- "S" is written next to the string literal `"(2, 3)"`.
- "returning" is written next to the `return` statement in the `PointV2.equals` method.
- "detemines that the PointV2 version of equals is invoked" is written in orange text.

# The equals Method: Overridden Version

Phase 4

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



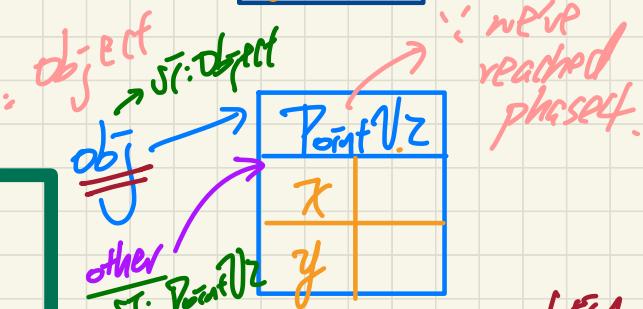
extends

DT of *this* (P.O.)

① *this* != *obj*  
② *obj* != null  
③ *this*.getClass() == *obj*.getClass()  
→ *obj*.getClass()

```
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;  
    }  
}
```

↓ ST: PointV2.



Possible Comparison

Complicated Error

*this*.x == *obj*.x  
&&  
*this*.y == *obj*.y

"x and  
y undefer.  
in ST of  
*obj*

# The `equals` Method: Overridden Version

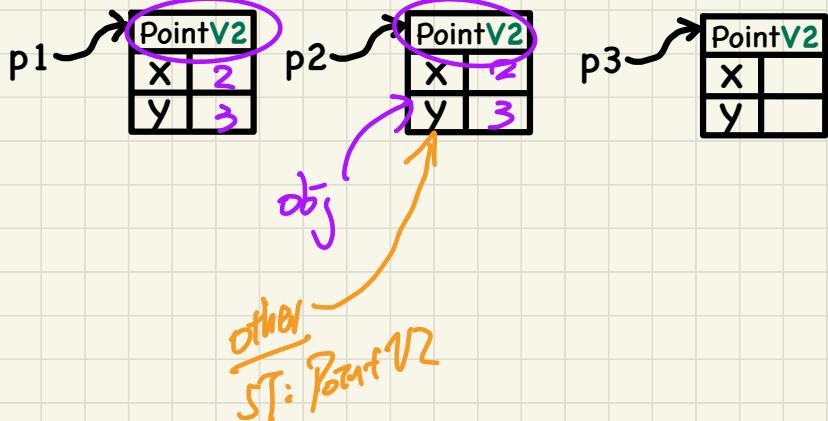
Example 1: Trace L10

```
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 }  
        PointV2 other = (PointV2) obj;  
        return this.x == other.x  
            && this.y == other.y;  
    }  
}
```

```
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); /* [REDACTED] */  
6 System.out.println(p2 == p3); /* [REDACTED] */  
7 System.out.println(p1.equals(p1)); /* [REDACTED] */  
8 System.out.println(p1.equals(null)); /* [REDACTED] */  
9 System.out.println(p1.equals(s)); /* [REDACTED] */  
10 System.out.println(p1.equals(p2)); /* [REDACTED] */  
11 System.out.println(p2.equals(p3)); /* [REDACTED] */
```



# The `equals` Method: Overridden Version

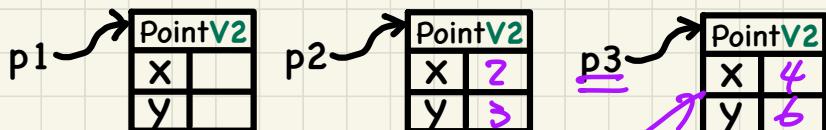
Example 1: Trace L11

```
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 }  
        PointV2 other = (PointV2) obj;  
        return this.x == other.x  
            && this.y == other.y;  
    }  
}
```

```
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); /* [REDACTED] */  
6 System.out.println(p2 == p3); /* [REDACTED] */  
7 System.out.println(p1.equals(p1)); /* [REDACTED] */  
8 System.out.println(p1.equals(null)); /* [REDACTED] */  
9 System.out.println(p1.equals(s)); /* [REDACTED] */  
10 System.out.println(p1.equals(p2)); /* [REDACTED] */  
11 System.out.println(p2.equals(p3)); /* [REDACTED] */
```



obj  
other  
ST: PointV2

Math

$$\begin{array}{ccc} P & \stackrel{\checkmark}{=} & Q \\ P & \stackrel{?}{=} & Q \end{array} \quad ) \quad \text{commutativity}$$
$$P \wedge Q = Q \wedge P$$

Java

$$\begin{array}{ccc} P & \cancel{\equiv} & Q \\ P & \equiv & Q \end{array} \quad \rightarrow \quad \text{short-circuit evaluation.}$$

# Short-Circuit Evaluation: &&

Left Operand op1	Right Operand op2	op1	&&	op2
true	true	true		
true	false	false		
false	true	false		
false	false	false		

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

guarding cond.  
↳ if it's false  
↳  $x = 0$   
↳ do not eval RHS  
(a) if el is false  
b) if el is true  
↳ eval e1 && e2  
↳ el && e2 → false

## Test Inputs:

x = 0, y = 10

x = 5, y = 10

1. Eval from L to R
2. To evaluate e1 && e2:
  - (1) Eval e1  
↳ if el is false  
b) if el is true  
↳ eval e1 && e2  
↳ el && e2 → false