Lecture 13 - June 24

Object Equality

Static Type, Dynamic Type, Type Casting
equals: Overriden 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

Static Type, Dynamic Type, Type Casting

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

may be changed ⟷ runtime type : determines the version of method called.

fixed ↑ declared type ↓ determines the range of methods that can be called on a variable

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

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

```
C1 o1 = new C1();
C2 o2 = new C2();
o1.m1();✓
o1.m2();✗
o2.m1();✗
o2.m2();✓
Object o3;
o3 = o1;
o3.m1;✗
o3.m2;✗
((C1) o3).m1();✓
((C1) o3).m2();✗
o3 = o2;
((C2) o3).m1();✗
((C2) o3).m2();✓
```

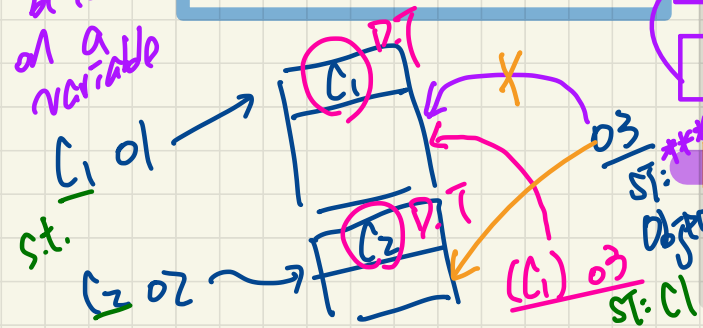
s.t. C1 d.t. C1 s.t. C2

m1 ded. in o1's d.t. ∵ m2 undef. ST in o1's ST

o3's ST remains object

the type cast (C1) o3 has ST: C1

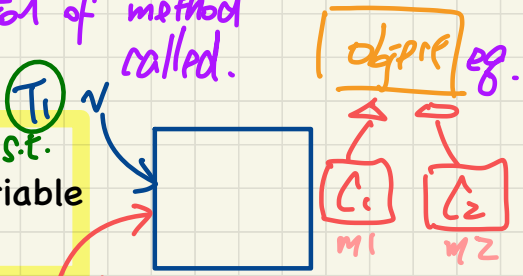
areas of ST: C2



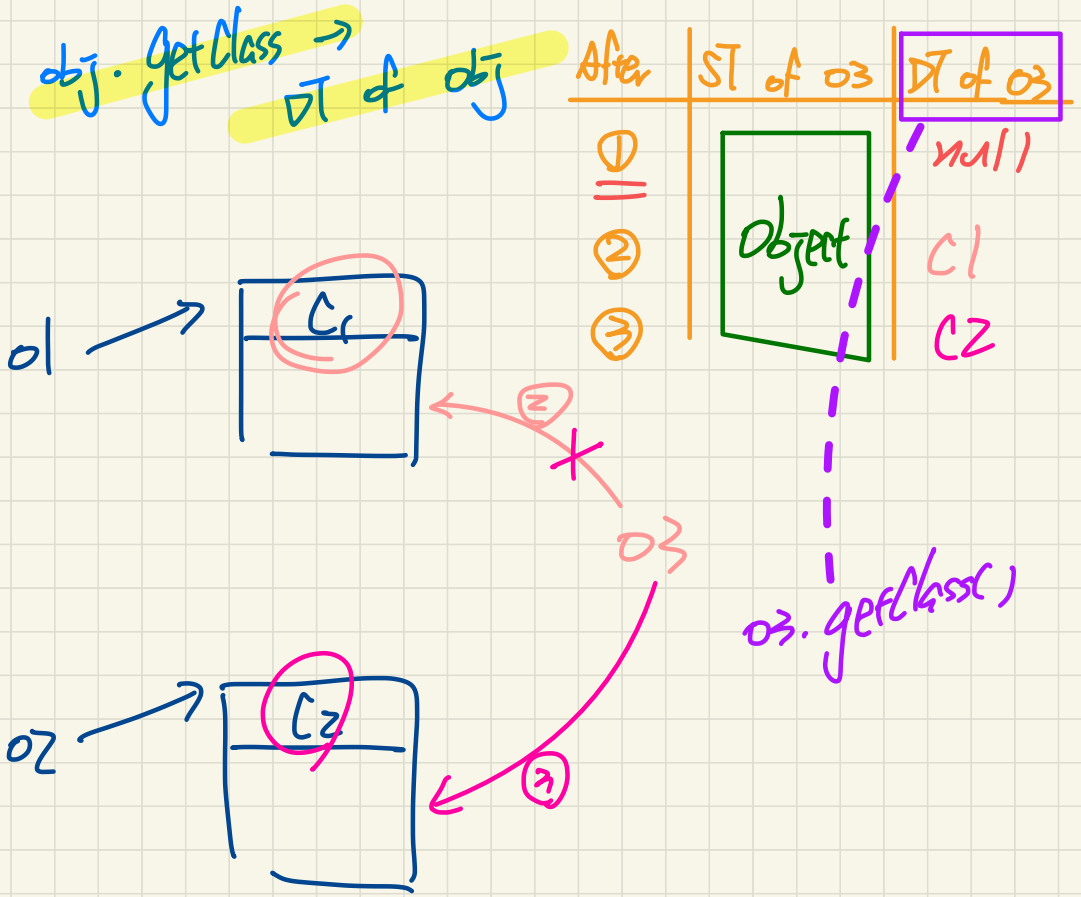
(T2) v
type cast of v ⇒ points to the same object but with a diff. s.t.

* o3 has ST: Object.
↳ o3.equals ✓
↳ o3.m1 ✗
↳ o3.m2 ✗

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;
   o3.m1;
   o3.m2;
   ((C1) o3).m1();
   ((C1) o3).m2();
③ o3 = o2;
   ((C2) o3).m1();
   ((C2) o3).m2();
```



The equals Method: Overridden Version

Phase 3

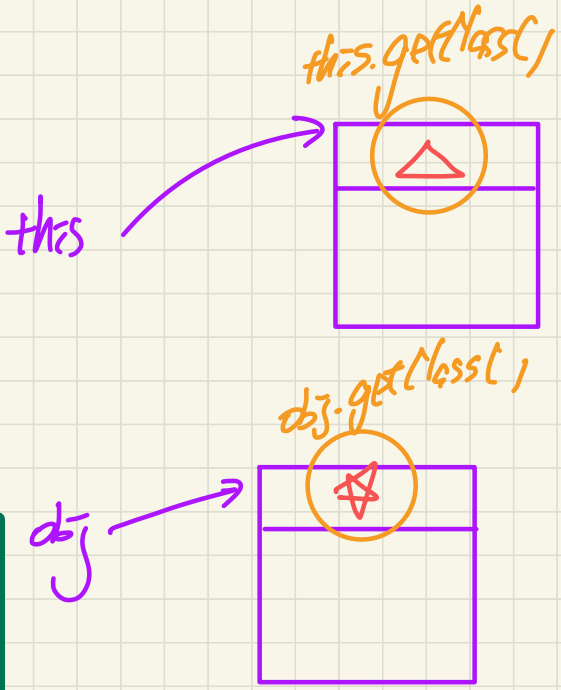
```
public class Object {
   ...
   public boolean equals(Object obj) {
      return this == obj;
   }
}
```

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

objects with different DTs are trivially not equal to each other



PointV2	
x	
y	

The equals Method: Overridden Version

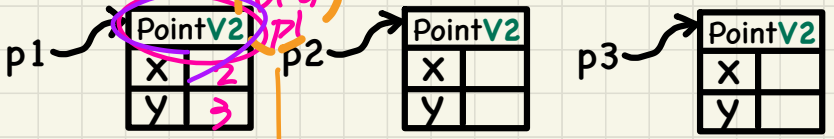
Example 1: Trace L9

```
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;
  }
}
```

```
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] */
```



call by value

obj = s

p1

determines that the PointV2 version of equals is invoked

s → "(2, 3)"

obj

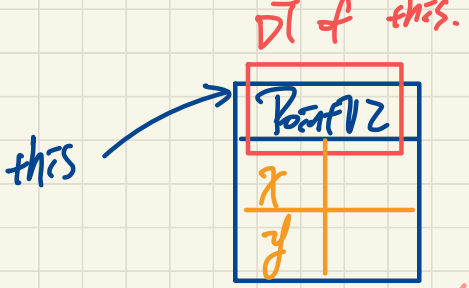
The equals Method: Overridden Version

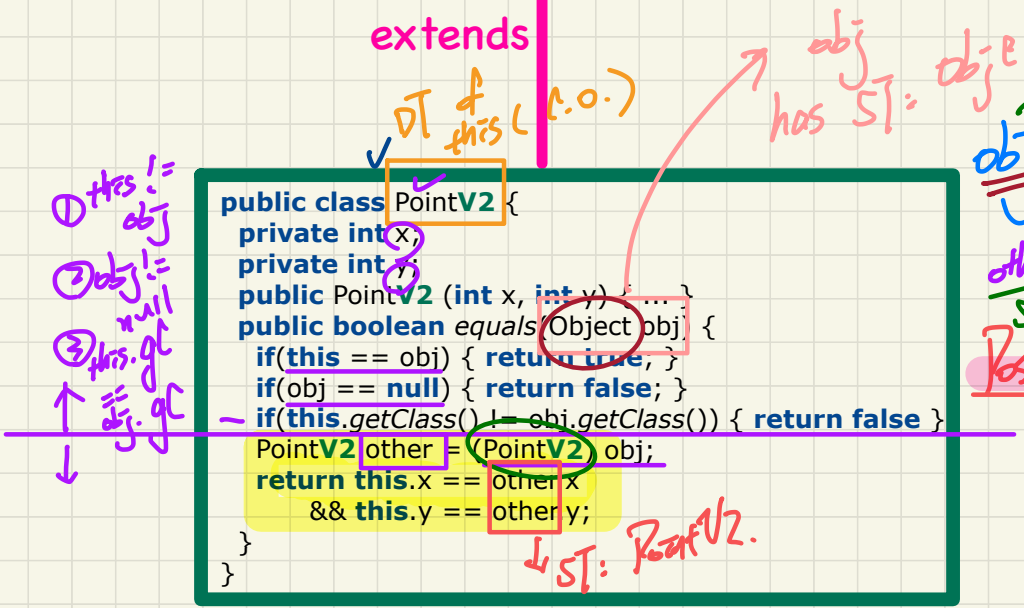
Phase 4

```
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;
  }
}
```





Possible Comparison

Compilation Error

this.x == obj.x

&&

this.y == obj.y

x and y undef. in ST of obj

The equals Method: Overridden Version

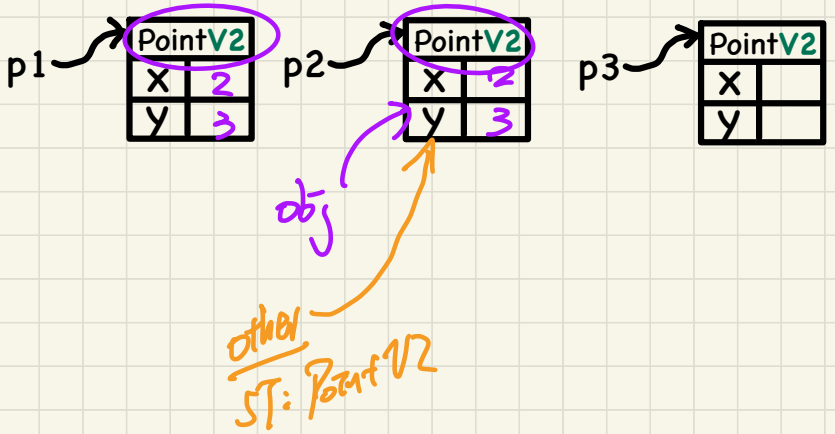
Example 1: Trace L10

```
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

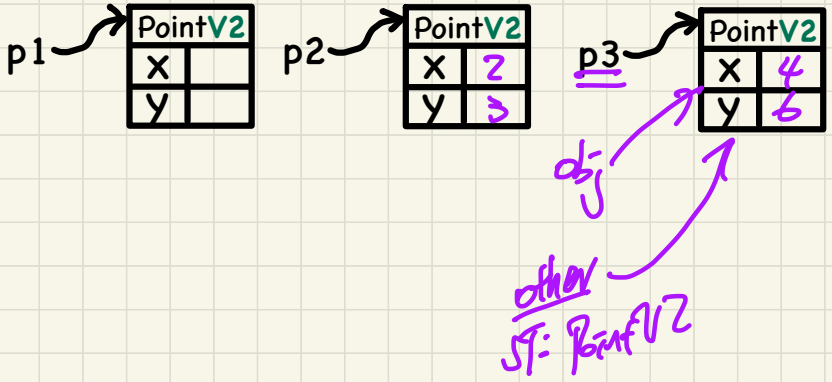
Example 1: Trace L11

```
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] */
```



Math

$p \wedge q$

$p \vee q$

Commutativity

$p \wedge q \equiv q \wedge p$

Java

p && q

p || q

→ 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

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) {
    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 the RHS

1. Eval from L to R
2. To evaluate e1 && e2:
   (1) Eval e1
   (a) if e1 is false
   ↳ no need to eval e2
   ↳ e1 && e2 → false
   (b) if e1 is true
   ↳ eval e2 to decide