Lecture 13 - Oct 23

Object Equality

Call by Value: Array Indexing
Deciding the Version of equals Method
Short-Circuit Evaluation

Announcements/Reminders

- Today's class: notes template posted
- WrittenTest1 next Wednesday (October 29)
  + Some practice questions on eClass
  + Past review session materials released
  + Lecture 12 and Lecture 13 this week are covered.
  + Zoom Review Session: 4 PM on Saturday, October 25
    * We can focus on revisiting parts of the covered topics.
- Lab3 released

Call by Value: Array Indexing

```
class Student {
  String[] courses; int noc;
  void register(String course) { ... }
}
```

```
RygSystem sys = new RygSystem();
Student s1 = new Student("alan");
Student s2 = new Student("mark");
sys.add(s1); sys.add(s2);
sys.register(s1, "EECS1090");
sys.register(s2, "MATH1025");
sys.registerAll("EECS2030");
```

RygSystem sys = new RygSystem()

```
class RegSystem {
  Student[] ss; int nos; ...
  void register(Student s, String course) {
    s.register(course);
  }
  void registerAll(String course) {
    for(int i = 0; i < nos; i ++) {
      this.register(this.ss[i], course);
    }
  }
}
```

c.b.v.

s = s1;

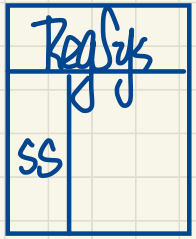
s = s2;

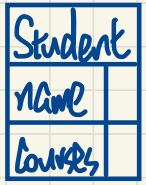
s = this.ss[i]

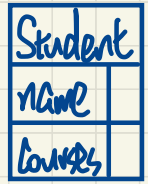
c.b.v.

course = "EECS2030"

c.b.v. happens for every iteration.







Method Call: When Multiple Versions are Available

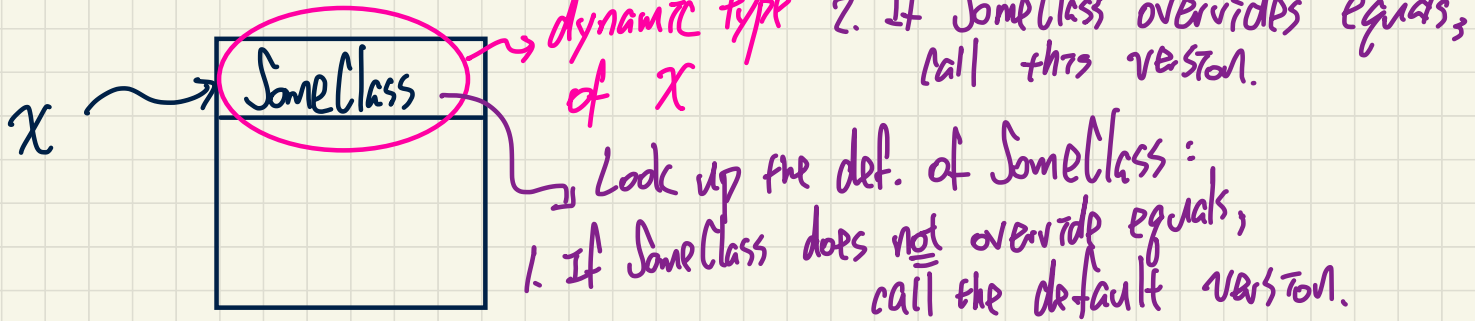
C.O. is all that matters to determine the version of equals method being called

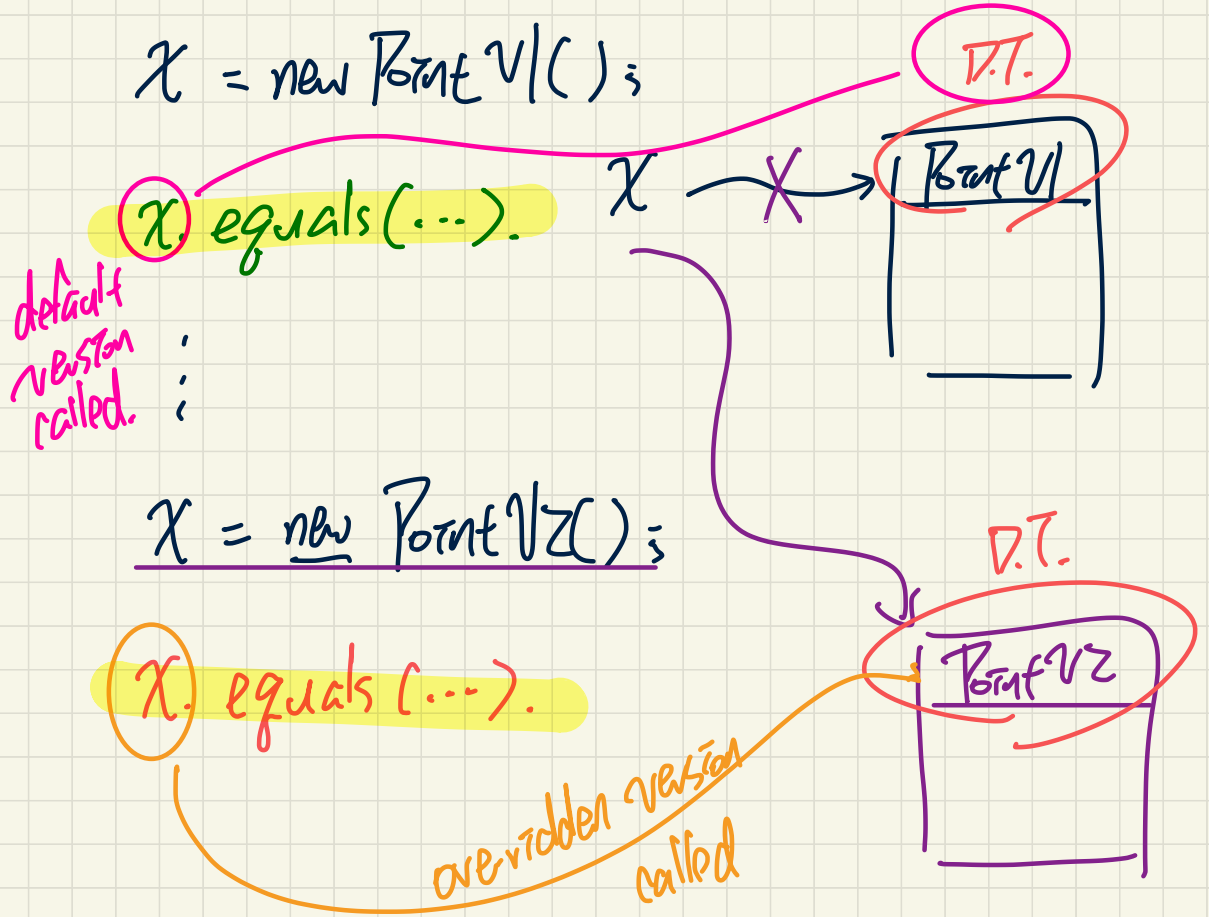
`x.equals(y)`

which version?
1. default version (from Object class)
2. overridden version (if exists).

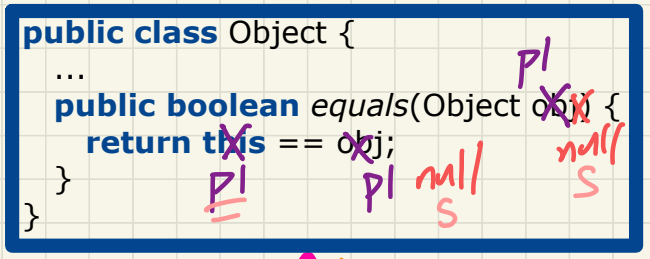
Principle:

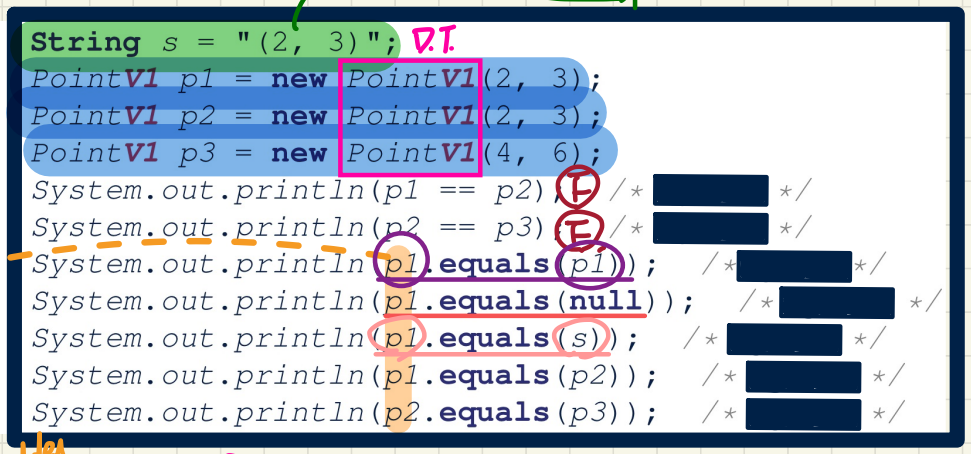
+ **Trace** the **dynamic type** of object "pointed to" by **reference variable x** at the point of executing this line of method call.
+ Is the **equals** method **overridden** in this **dynamic type**?
  * Yes -> Invoke the **overridden** version.
  * No -> Invoke the **default version** (inherited from **Object**).



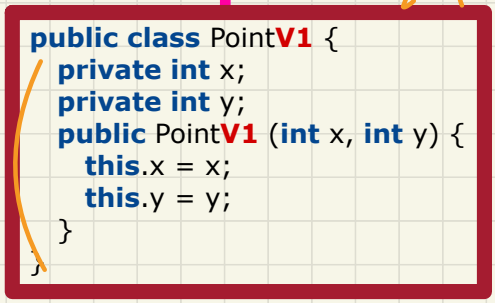

x = new Point V1();
x.equals(...).
default version called.
x
D.T.
Point V1
x = new Point V2();
x.equals(...).
overridden version called
D.T.
Point V2

The equals Method: Default Version

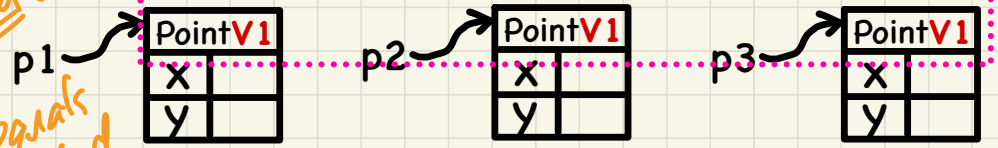




extends



dynamic type



27: p1 == p1 → T

28: p1 == null → F.

29: "p1 == s" → F.

PointV1 p1 = new PointV1 . . . ;

String s = "..." ;

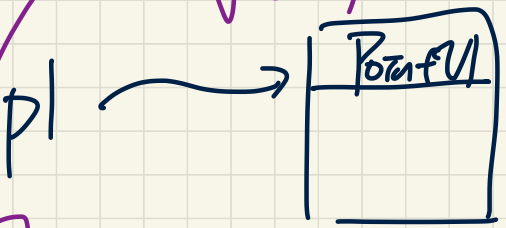
declared types different ↳ forbidden to compare their addresses directly.

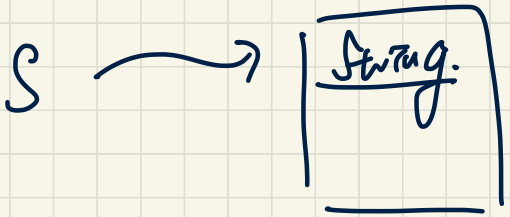
① p1.equals(s); → F.

↳ "p1 == s" → Compiles ; boolean equals(Object obj)

② p1 == s → compilation error

any type of argument is acceptable.





Call by Value: Invoking the Overridden equals

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

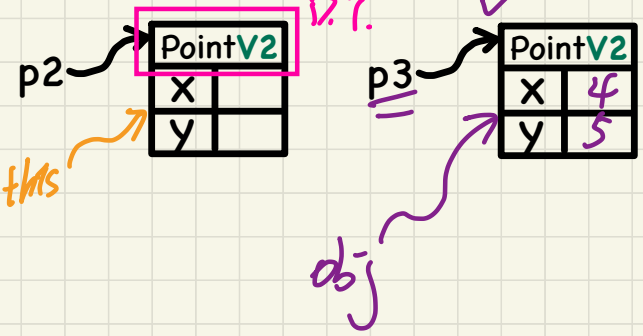
extends

```
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;
}
```

C.b.v. obj = p3

```
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);   /* true */
5  System.out.println(p1.equals(p1));   /* true */
6  System.out.println(p1 == p2);   /* false */
7  System.out.println(p1.equals(p2));   /* true */
8  System.out.println(p2 == p3);   /* false */
9  System.out.println(p2.equals(p3));   /* false */
```

P.T.



Short-Circuit Evaluation: && and, conjunction

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

2.1

the only way for && to be T, is when both operands are true.

Test Inputs:

x = 0, y = 10

x = 5, y = 10

op1 && op2

```
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");
    }
}
```

Division by zero. if x == 0

(F) 0 != 0 && 10/0 > 2 — skipped

(T) 5 != 0 && 10/5 > 2

1. Evaluate L to R

2.1. If op1 eval. to T
↳ eval op2 to decide the final result of &&

2.2. If op1 eval. to F
↳ skip eval. of op2 (F) the final result of && is

Short-Circuit Evaluation: || (or, disjunction)

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

2.1

2.2.

the only way to make || (F) is both operands are false.

Test Inputs:

x = 0, y = 10

x = 5, y = 10

op1 || op2

```
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");
}
```

1. Evaluate L to R

2.1. If op1 eval. to F
↳ eval op2 to decide the final result of ||

2.2. If op1 eval. to T
↳ skip eval. of op2 the final result of || is (T)

Math

$p \wedge q$

$p \vee q$

Commutativity

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

Java

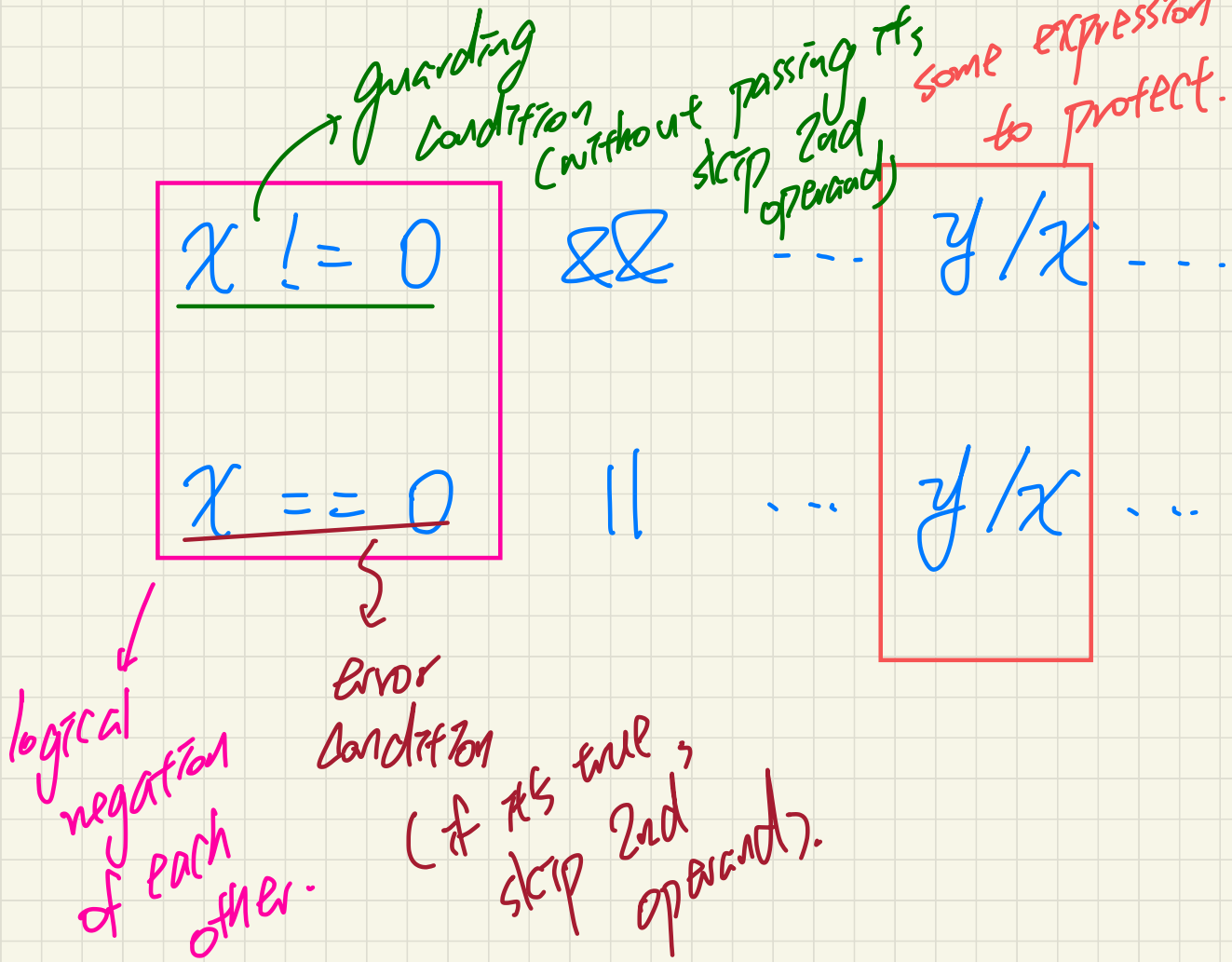
$p \;\&\&\; q$

$p \;||\; q$

$\rightarrow$ $p \;\&\&\; q$ / $q \;\&\&\; p$

not identical

↳ short-circuit evaluation


guarding condition
without passing it's 2nd operand
some expression to protect.
x != 0
&&
...
y/x ...
x == 0
||
...
y/x ...
logical negation of each other.
error condition
(if it's true, skip 2nd operand).

a: array of values

a.length == 10

0..9

is this always guarded?

No.

T

10

i != 0 && a[i] == v || && i <= a.length - 1

g1 g2

i == length of array (10)

10

a[10] → IOOBE!

Correct: g1 && g2 && ... a[i]

Exercise: use ||

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 */ }
```

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

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