Lecture 13 - Oct 22

Object Equality

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

Method Call: When Multiple Versions are Available

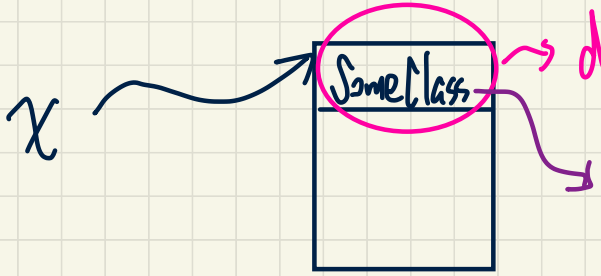
C.O. is all that matters to determine the version of equals to be invoked.

`x.equals(y)`

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

Principle:

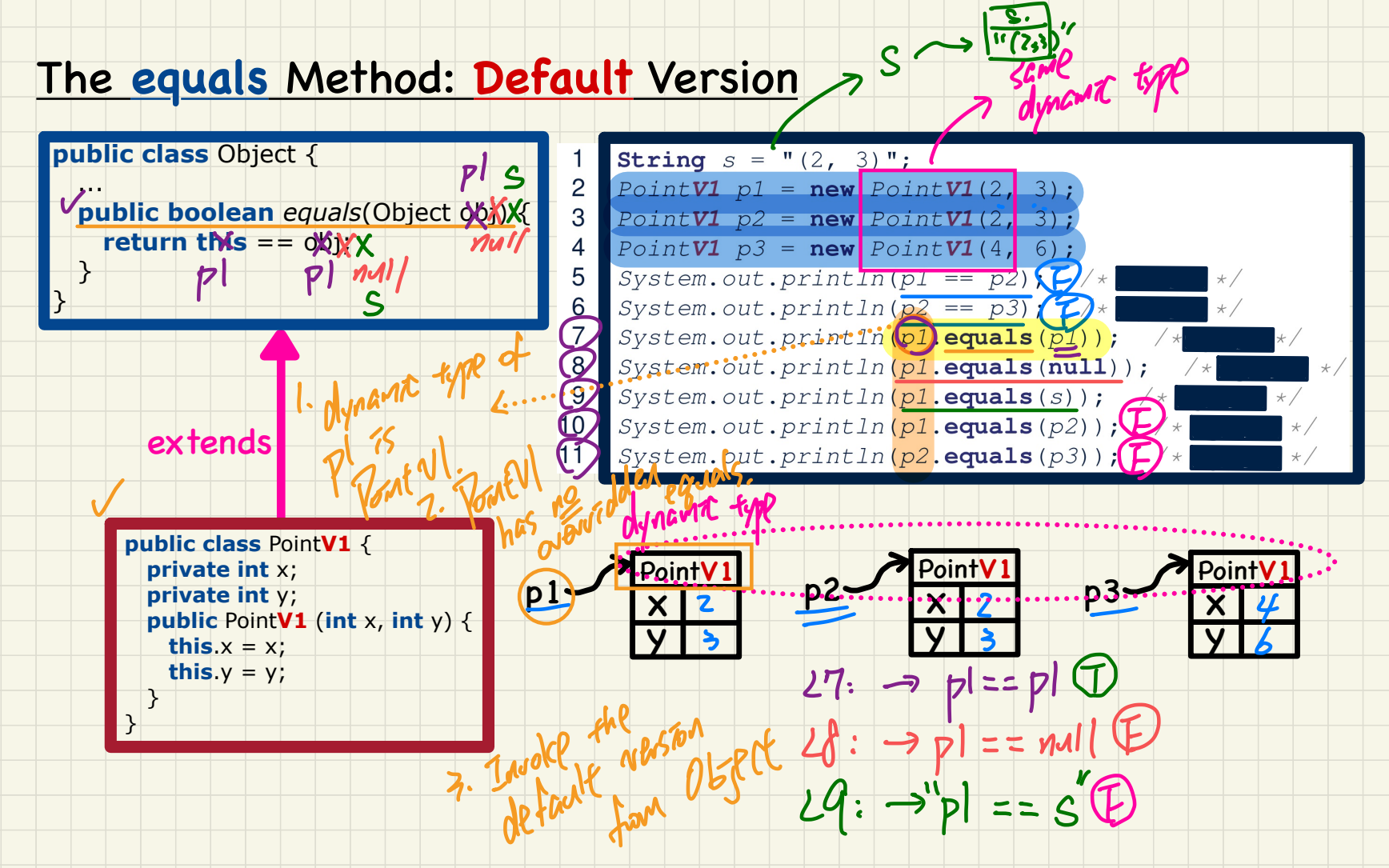
+ **Trace** the **dynamic type** of object "pointed to" by **reference variable x** at the point of executing this line of method call.
+ Is the **equals** method **overridden** in this **dynamic type**?
  * Yes -> Invoke the **overridden** version.
  * No -> Invoke the **default version** (inherited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



dynamic type of x

1. If SomeClass has an overridden equals, invoke that version.
2. Otherwise, invoke the default version (from Object).

The equals Method: Default Version



Point V1 p1 = new ... ;

String s = "..." ;

String s1 = new String("a"); (D.T.)

String s2 = new String("a");

s1 == s2 (F.)

s1.equals(s2) (T.)

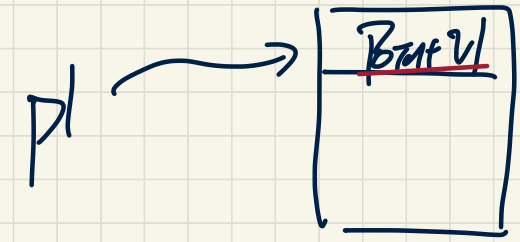
① p1.equals(s) → (F)

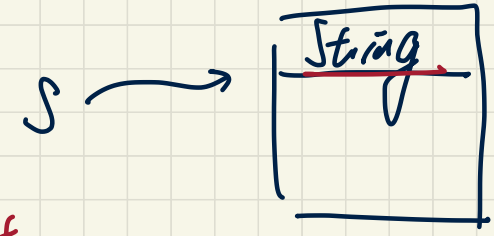
↳ "p1 == s"

compile: argument of equals is simply expecting an object.

② p1 == s → compilation error!

∵ comparing addresses of two different types





Call by Value: Invoking the Overridden equals

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

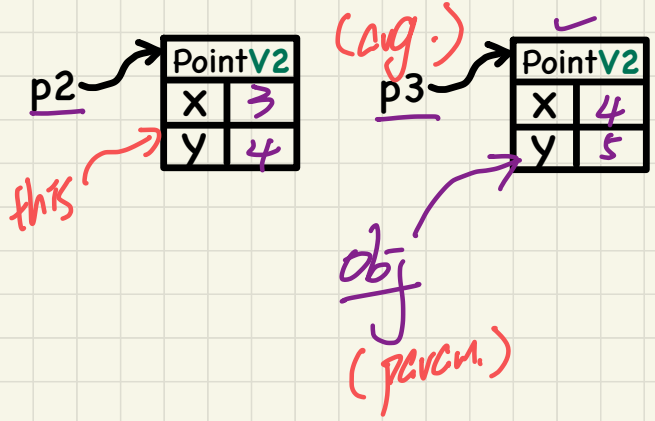
extends

```
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;
  }
}
```

overridden version

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 */
```



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

2.2

→ if at least one operand is false, && is false op1 && op2

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");
    }
}
```

if x == 0 → DivisionByZeroException.

expression that should not fail to eval.

0 != 0 (F.) && 10/0 > 2 (F.) skipped

5 != 0 && 10/5 > 2

(T) 2 > 2

1. Evaluation is from L to R.

2.1 If op1 eval. to T, eval op2, and its result is the final result

2.2. If op1 eval. to F, skip eval. of op2 → overall && is (F.)

DBZE if $x == 0$

$x != 0$ && $y / x > 2$

guarding condition

RHS

↳ 1. If $x != 0$ eval. to T,
Continue to eval. y/x

2. If $x != 0$ eval to F → $x == 0$
↳ skip the eval. of RHS.

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

2.2.
2.1

if at least one operand is true, || is true.

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");
}
```

T `0 == 0` || `10/0 > 2` (skip.)

F `5 == 0` || `10/5 > 2` (10/5 > 2)

op1 || op2

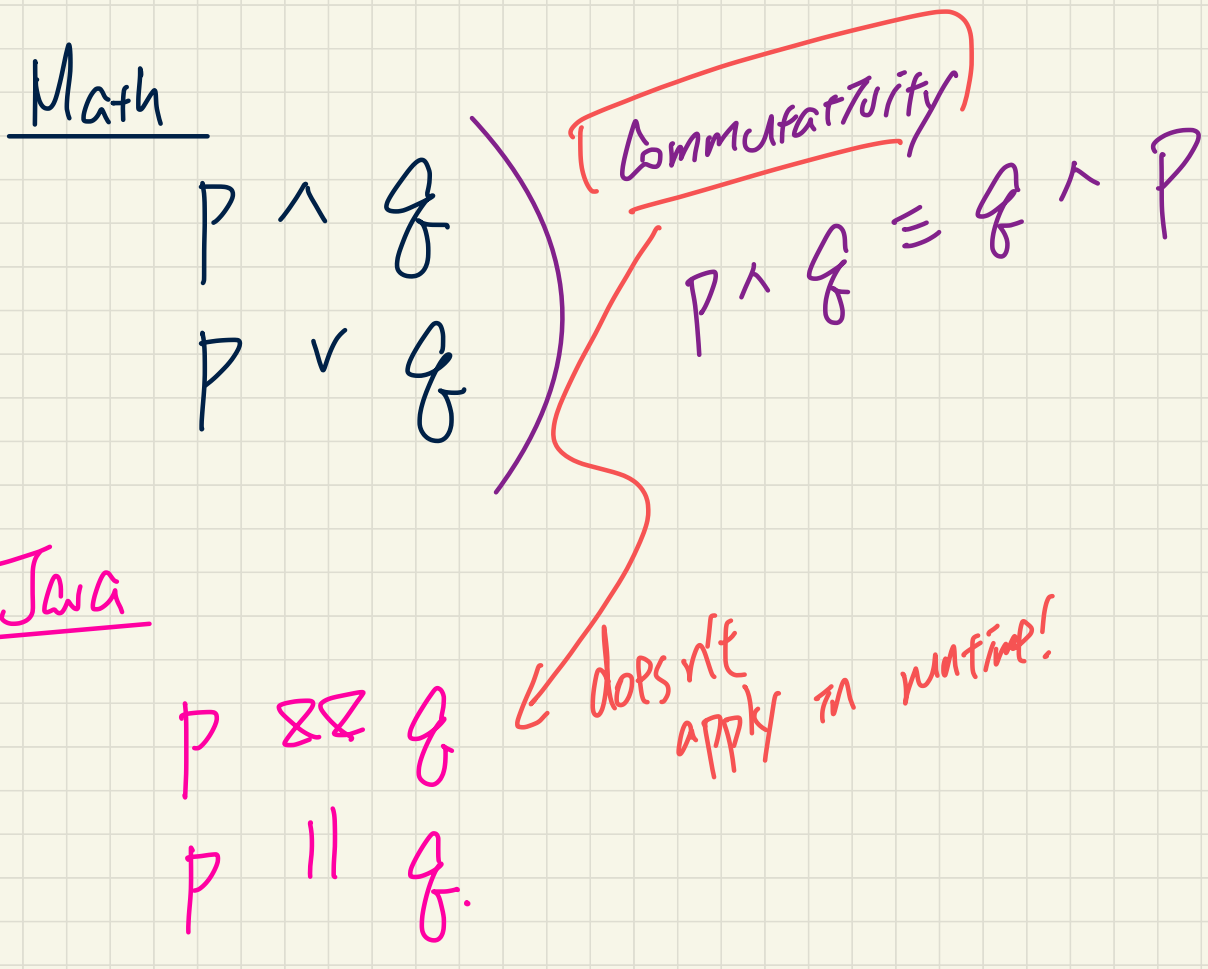
1. Eval: L to R

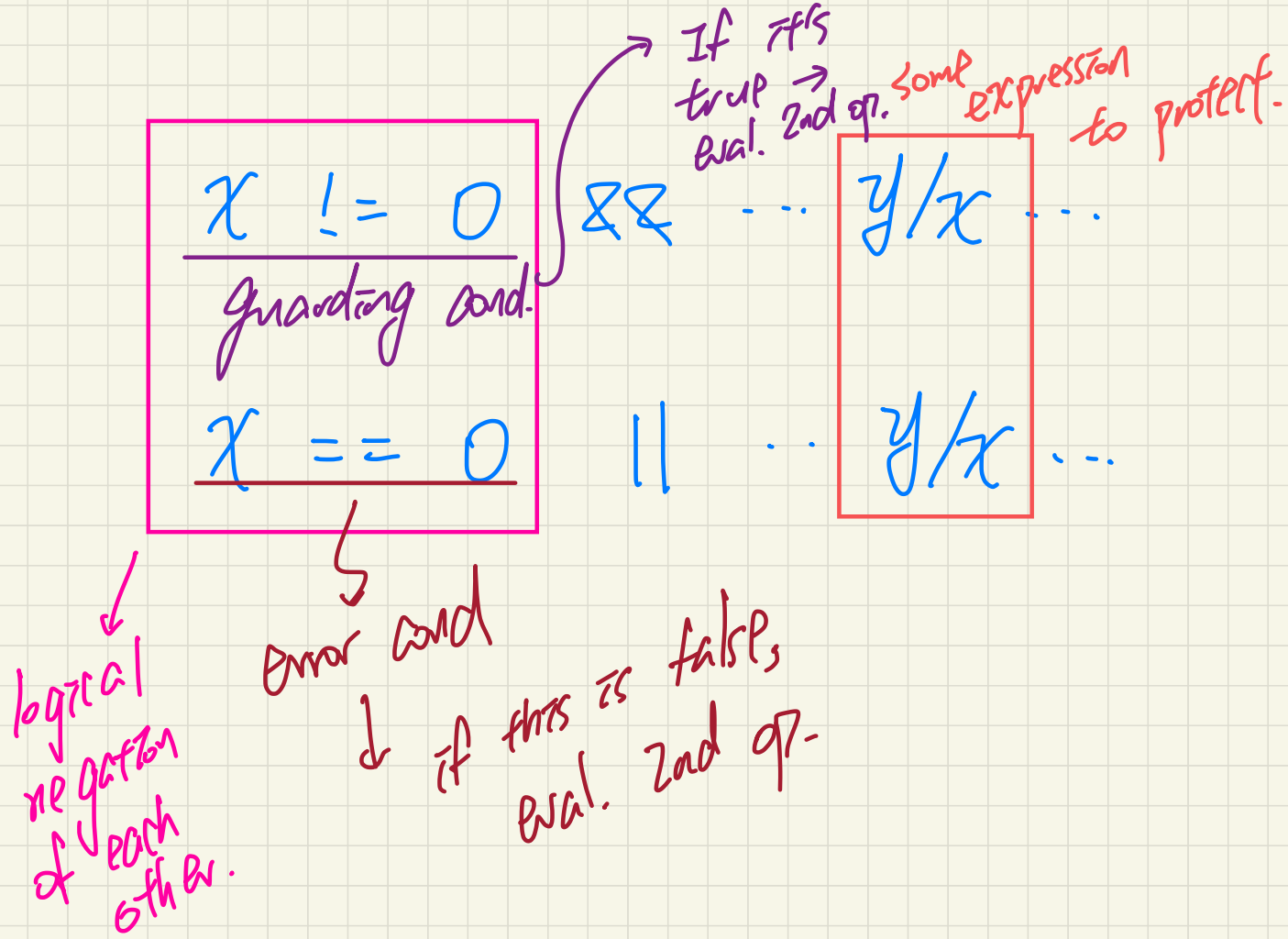
2.1 If eval. op1 is T

skip eval. of op2, overall result of || is T.

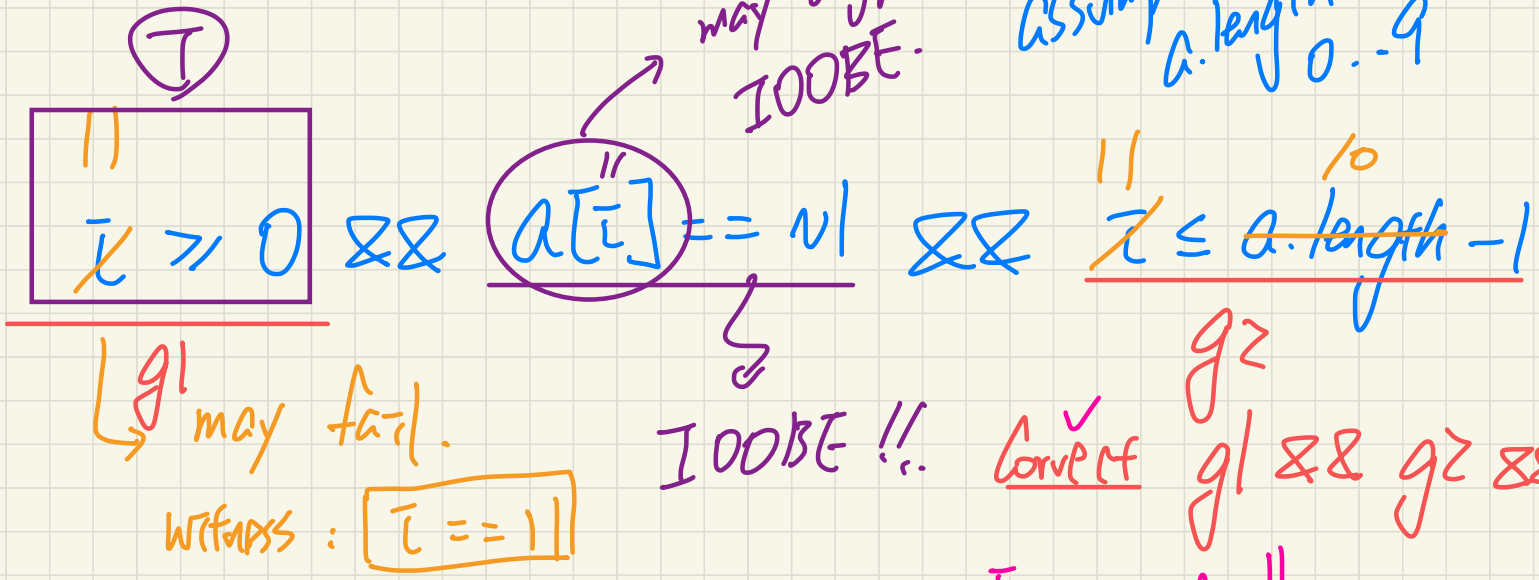
2.2 If eval. op1 is F

eval. op2, and its result is the overall result of ||.


Math
p ∧ q
p ∨ q
Commutativity
p ∧ q ≡ q ∧ p
Java
p && q
p || q.
does not apply in runtime!


x != 0
guarding cond.
x == 0
&&
||
y/x
y/x
If this true → eval. 2nd op.
some expression to protect.
logical negation of each other.
error cond
↓ if this is false, eval. 2nd op.

a : array of values



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 */ }
```