Lecture 13 - Oct. 24

Object Equality, Call by Value

Short-Circuit Evaluation: && vs. ||
equals: Person vs. PersonCollector

Announcements/Reminders

- Lab2 due tomorrow at noon
- Lab3 to be released tomorrow
- ProgTest2 next Wednesday, October 30
  + PDF Guide released

assertEquals: Reference Comparison or Not

* Writing p2 == p3 directly causes compilation error.

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);
```

p1.eq(p2) → p1 == p2 → False (fail)

p2.eq(p3) → "p2 == p3" → False (fail)

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);
① assertEquals(p2, p3);
② assertEquals(p3, p2);
```

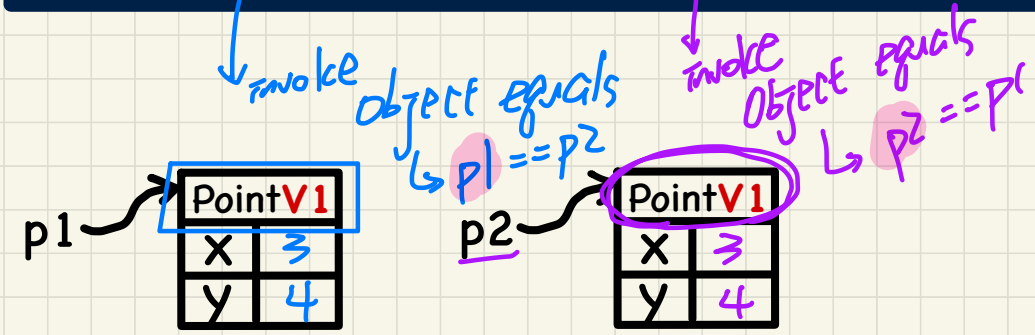
D.T. PointV2 → PointV2 ver.

D.T. PointV1 → Object ver.

which version of equals invoked?

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());
}
```




```
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, p3); */ /* both fail */
  assertTrue(p3.equals(p4)); assertTrue(p4.equals(p3));
  assertEquals(p3, p4); assertEquals(p4, p3);
}
```

pass F

assertFalse(p3 == p4)

fail

assertSame(p3, p4)

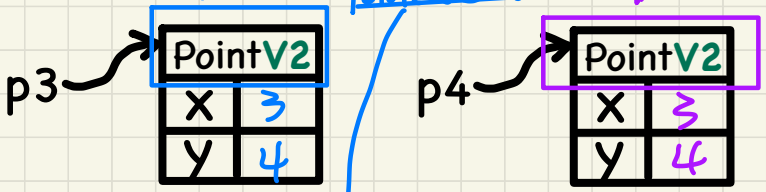
p3 == p4

↓

false

```
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;
  }
}
```

true

Testing Equality of Points in JUnit: Default vs. Overridden

```
@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); -> "p1==p2" -> fail */
  /* assertSame(p2, p1); -> "p2==p1" -> fail */
  /* version of equals from Object is called */
  assertFalse(p1.equals(p2)); -> pass
  /* version of equals from PointP2 is called */
  assertFalse(p2.equals(p1)); -> pass
}
```

① ② * **

equals ↳ fake

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

D.T.

p1 → PointV1 { x: 3, y: 4 }

D.T.

p2 → PointV2 { x: 3, y: 4 }

* p1.equals(p2)
↳ Object ver.
↳ p1 == p2
↳ F

extends

** p2.eq.(p1)
↳ PointV2 ver.

extends

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

```
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;
  }
}
```

PV1

PV2

Math

$p \wedge q$

$p \vee q$

Commutativity

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

Java

p && q

p || q

p && q (short-circuit evaluation!)

q && p

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");
    }
}
```

0 != 0 && 10/0 > 2

F. F bypass.

g.c.

exp1 && exp2

① Eval. L → R

② Eval exp1 : false
Skip eval. of exp2
&& eval to false.

② Eval exp1 : true
Eval exp2.

Short-Circuit Evaluation: ||

Left Operand op1	Right Operand op2	op1 || op2
false	false	false
true	false	true
false	true	true
true	true	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");
}
```

e.c.

0 == 0 || 10/0 > 2

T || bypass

return (T).

exp1 || exp2

① Eval exp1 → T

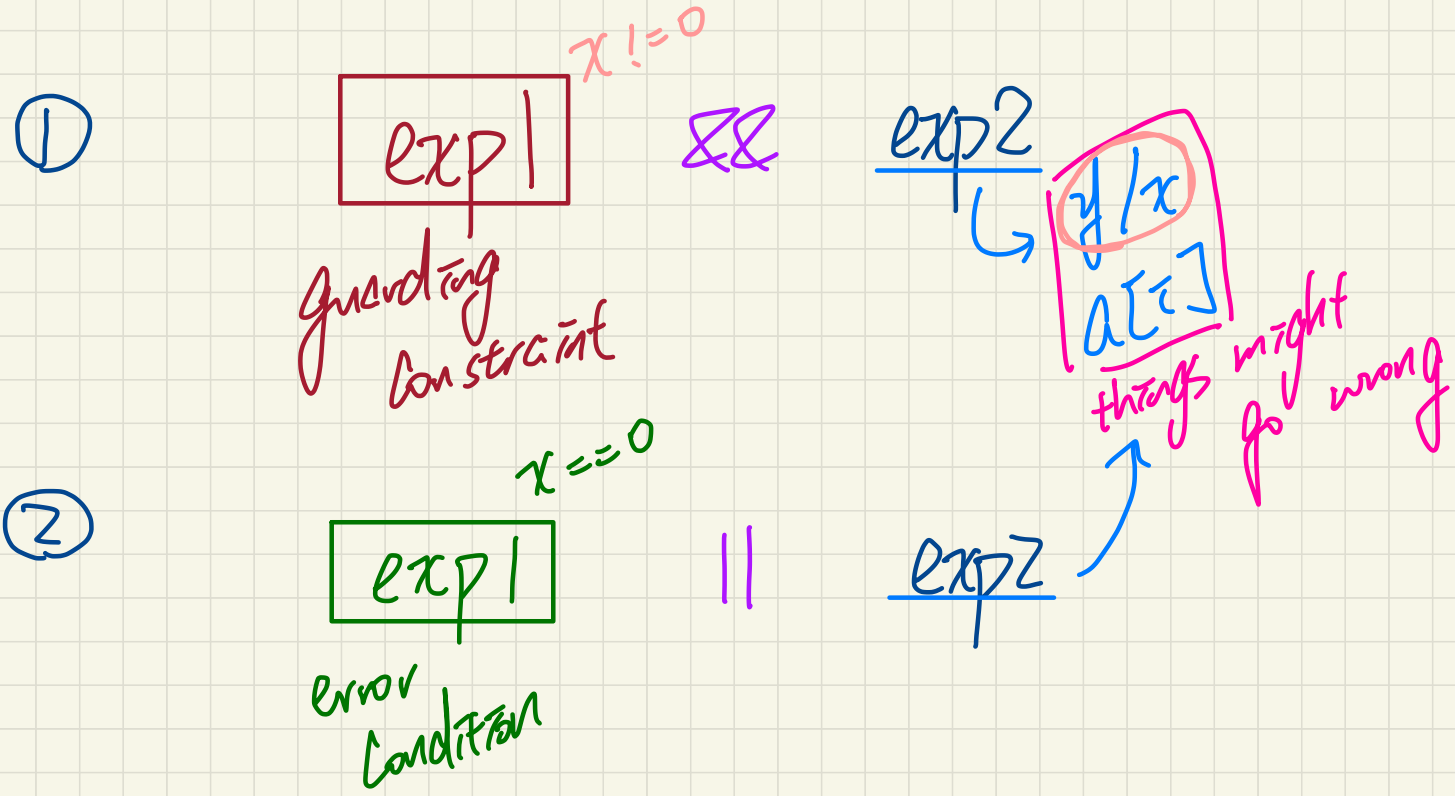
② Skip Eval. exp2

↳ _||_ return (T).

② Eval exp1 → F

Eval exp2.

Short Circuit Evaluation



Exercise

① (A) && (B) && a[i] > 0

AIDFBE

② (C) || (D) || a[i] > 0

Short-Circuit Evaluation: Common Errors

Exercise

Test Inputs:
x = 0, y = 10

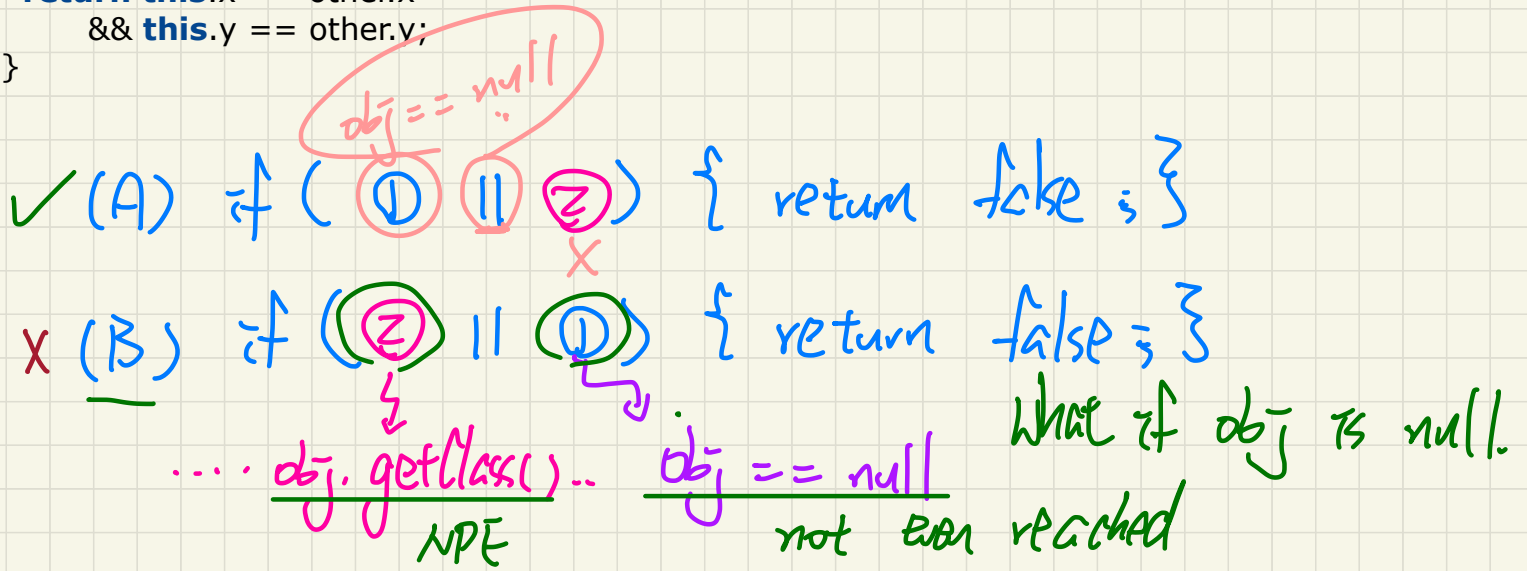
Short-Circuit Evaluation is not exploited: crash when `x == 0`

```
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 */ }
```

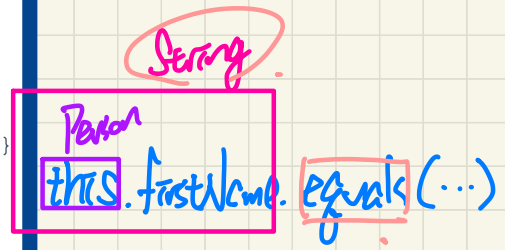
```java
public class PointV2 {
  private int x;
  private int y;
  public PointV2 (int x, int y) { ... }
  public boolean equals(Object obj) {
    if(this == obj) { return true; }
    if(obj == null) { return false; }
    if(this.getClass() != obj.getClass()) { return false }
    PointV2 other = (PointV2)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



Exercise: Two Persons are equal if their names and measures are equal

```
1  public class Person {
2    private String firstName; private String lastName;
3    private double weight; private double height;
4    public boolean equals(Object obj) {
5      if(this == obj) { return true; }
6      if(obj == null || this.getClass() != obj.getClass()) { return false; }
7      Person other = (Person) obj;
8      return
9          this.weight == other.weight
10       && this.height == other.height
11       && this.firstName.equals(other.firstName)
12       && this.lastName.equals(other.lastName);
13   }
14 }
```

(1) Object
(2) PointV2
(3) Person
(4) String



Q1: At Line 6, will there be a **NullPointerException** if obj == **null**?

Q2: At Line 6, what if we change it to:

if(**this**.*getClass*() != obj.*getClass*() || obj == **null**)

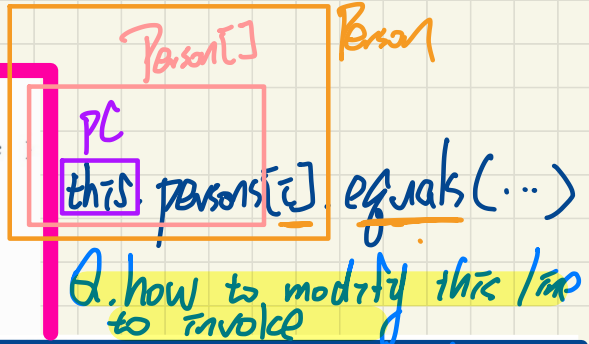
Q3: At Lines 11 & 12 which version of the **equals** method is called?

Exercise: PersonCollectors are equal if their arrays of persons are equal

```
class PersonCollector {
  private Person[] persons;
  private int nop; /* number of persons */
  public PersonCollector() { ... }
  public void addPerson(Person p) { ... }
  public int getNop() { return this.nop; }
  public Person[] getPersons() { ... }
}
```

```
1  public boolean equals(Object obj) {
2   if(this == obj) { return true; }
3   if(obj == null || this.getClass() != obj.getClass()) { return false; }
4    PersonCollector other = (PersonCollector) obj;
5   boolean equal = false;
6   if(this.nop == other.nop) {
7     equal = true;
8     for(int i = 0; equal && i < this.nop; i ++) {
9       equal = this.persons[i].equals(other.persons[i]);
10    }
11  }
12  return equal;
13 }
```

Q: At Line 9 of PersonCollector's equals method which version of the equals method is called?




```
1  public class Person {
2    private String firstName; private String lastName;
3    private double weight; private double height;
4    public boolean equals(Object obj) {
5      if(this == obj) { return true; }
6      if(obj == null || this.getClass() != obj.getClass()) { return false; }
7      Person other = (Person) obj;
8      return
9          this.weight == other.weight
10       && this.height == other.height
11       && this.firstName.equals(other.firstName)
12       && this.lastName.equals(other.lastName);
13   }
14 }
```