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

Short-Circuit Evaluation: &&

Conjunction

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

eval →

expl && exp2

```
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");
    }
}
```

x = 0, y = 10

0 != 0 && 10/0 > 2

F → false. skip

① expl : T

eval exp2

② expl : F

skip eval. exp2

→ && → F

Short-Circuit Evaluation: ||

disjunction

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");
}
```

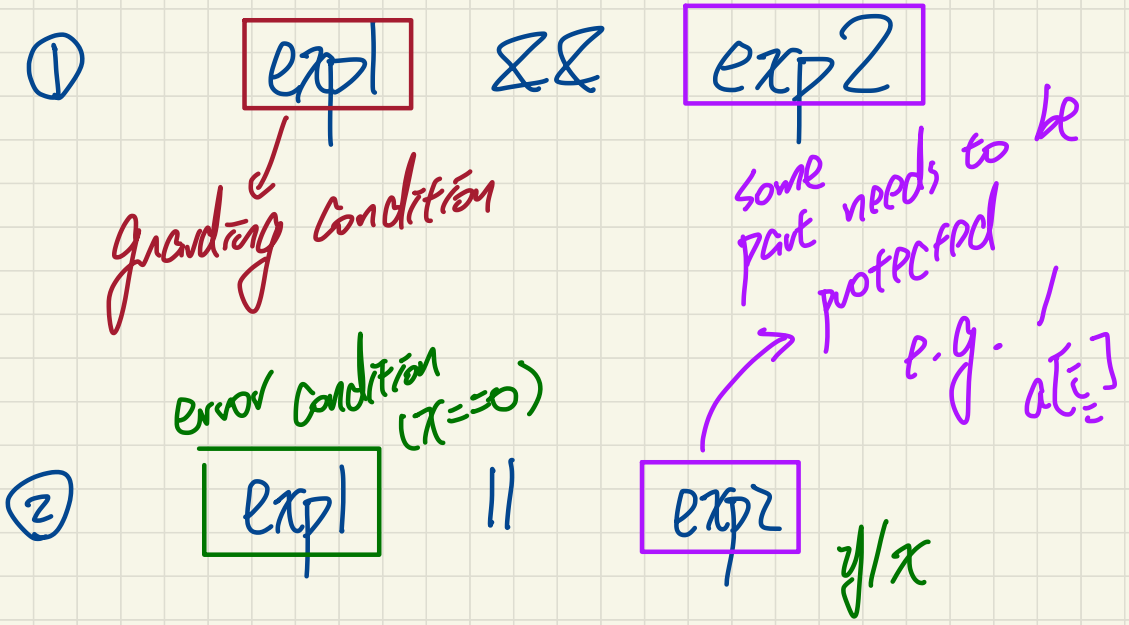
0 == 0 (T) || 10/0 > 2 skipped (T)

exp1 || exp2

① exp1 : F
eval. exp2 to decide

② exp1 : T ✓
skip eval. exp2
↳ _||_ T.

Short Circuit Evaluation



Exercise

① ____A.____ && ____B.____ && a[i] > 0

② ____C.____ || ____D.____ || a[i] > 0

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 */ }
```

|10/0| > 2 → crash.

↓ guarding cond. will not be reached!

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

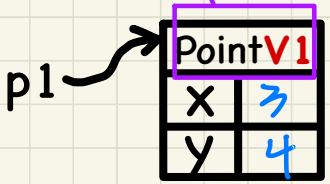
```
if (y / x <= 2 || x == 0) {
  /* print error */
}
else {
  /* do something */ }
```

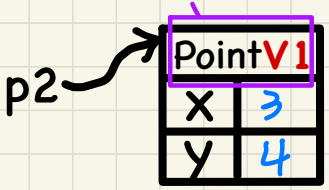
Testing Default Equality of Points in JUnit

```
@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());
}
```

p1 == p2

p2 == p1






```
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;
  }
}
```

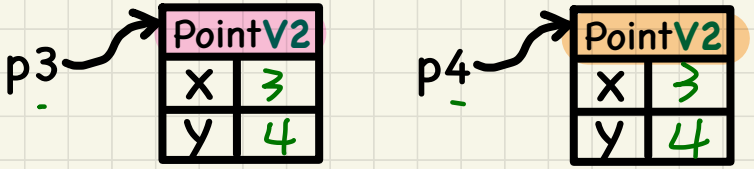
Testing Overridden Equality of Points in JUnit

```
@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 */
①assertTrue(p3.equals(p4)); assertTrue(p4.equals(p3));
②assertEquals(p3, p4); assertEquals(p4, p3);
}
```

```
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 assertTrue(p3.equals(p4))

② assertEquals(p3, p4)

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); */ /* compiles, but fails */
  /* assertSame(p2, p1); */ /* compiles, but fails */
  /* version of equals from Object is called */
  assertFalse(p1.equals(p2));
  /* version of equals from PointP2 is called */
  assertFalse(p2.equals(p1));
}
```

"p1==p2"

Q. Do these two assertions pass for the same reason?

* p1 == p2 (F)

** equals from PointV2

Q.

(F)

(F)

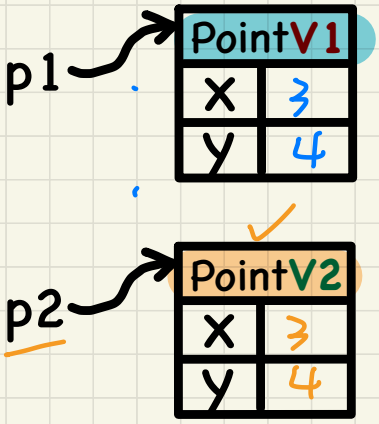
```
public class Object {
  ...
  public boolean equals(Object obj) {
    return this == obj;
  }
}
```

* D.T. of C.O. PointV1

** D.T. of C.O. PointV2

extends

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;
  }
}
```

*Exercise

Combine ① and ② using conjunction (&&).

```
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; }
①   if(obj == null) { return false; }
②   if(this.getClass() != obj.getClass()) { return false }
    PointV2 other = (PointV2) obj;
    return this.x == other.x
        && this.y == other.y;
  }
}
```

(A) if (① || ②) { return false; }

① → obj == null

avoids NPE when obj is null

(B) if (② || ①) { return false; }

② → ... obj.getClass()

① → obj == null

may still have NPE when obj is null

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 }
```

this.firstName.equals(...)

Person

String.

① Object ② Person ③ 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**)

NPE if obj is 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() { ... }
}
```

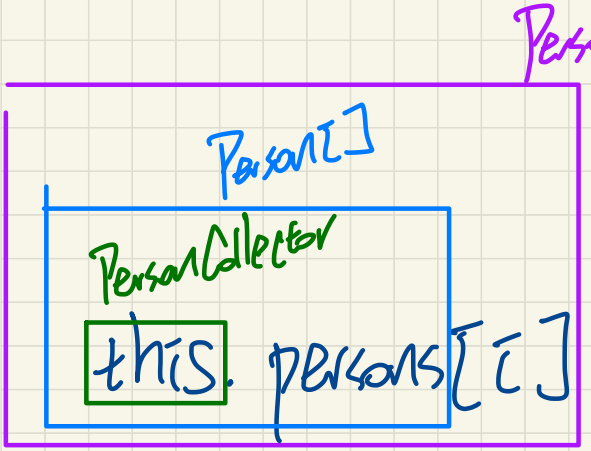
```
1  public boolean equals(Object obj) {
2    if(this == obj) { return true; }
3    if(obj == null || this.getClass() != obj.getClass()) { return false; }
4    PersonCollector other = (PersonCollector) obj;
5    boolean equal = false;
6    if(this.nop == other.nop) {
7      equal = true;
8      for(int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

as soon as equal becomes false exit from the loop

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 }
```

* the two Person[] store the same # of persons.



(A) Object

(B) Person

(C) PersonCollector

(D) String

Alt.

this.persons[i].getFN().equals(other.persons[i].getFN())

String class