# Lecture 12 - Oct. 22

## Object Equality

***Reference Equality vs. Object Equality***
***JUnit: assertSame vs. assertEquals***
***Logical Op.: Short-Circuit Evaluation***

## Announcements/Reminders

- ProgTest1 results to be released by next Monday
- Lab2 due this Friday
- ProgTest2 on Wednesday, October 30

  + PDF Guide released

# 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;
  }
}
```

1. this != obj
2. obj != null
3. this and obj have same RT.

PointV2.

static type

can invoke only "obj" methods from Object class

compiles ! ST of other is PointV2

this, other → PointV2 (ST: PV2)

obj → PointV2 (ST: Object)

* without this line of type casting,

return this.x == obj.x
&& this.y == obj.y

ST: Object

⇒ x and y are not applicable

| PointV2 | |
|---|---|
| x | |
| y | |

# The equals Method: Overridden Version

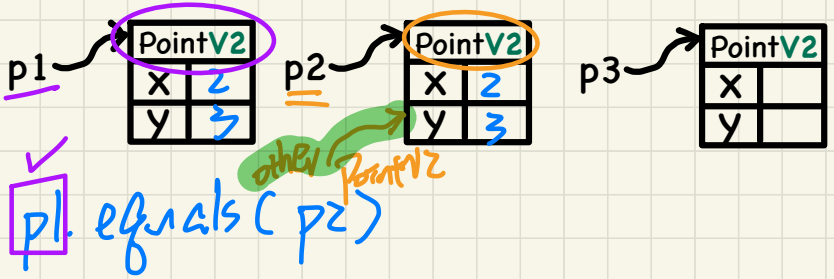
Example 1: Trace L10

```
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);   /*        */
6  System.out.println(p2 == p3);   /*        */
7  System.out.println(p1.equals(p1));   /*        */
8  System.out.println(p1.equals(null));   /*        */
9  System.out.println(p1.equals(s));   /*        */
10 System.out.println(p1.equals(p2));   /*        */
11 System.out.println(p2.equals(p3));   /*        */
```



# The equals Method: Overridden Version

Example 1: Trace L11

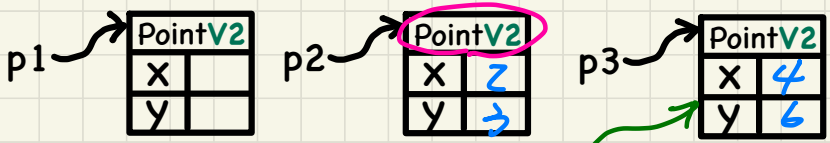
```
public class Object {
   ...
   public boolean equals(Object obj) {
      return this == obj;
   }
}
```

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);   /*       */
6  System.out.println(p2 == p3);   /*       */
7  System.out.println(p1.equals(p1));   /*       */
8  System.out.println(p1.equals(null));   /*       */
9  System.out.println(p1.equals(s));   /*       */
10 System.out.println(p1.equals(p2));   /*       */
11 System.out.println(p2.equals(p3));   /*       */
```

F.




p2.equals(p3)

C.O.
D.T.: PointV2

other
ST: PointV2

# The equals Method:

# To Override or Not?

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

extends

extends

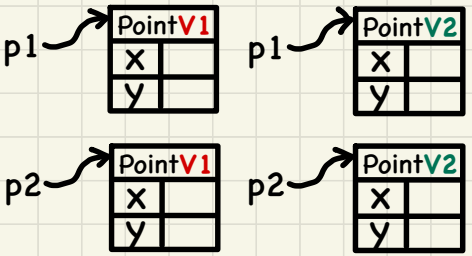
```
public class PointV1 {
   private int x;
   private int y;
   public PointV1 (int x, int y) {
      this.x = x;
      this.y = y;
   }
}
```

```
public class PointV2 {
   private int x; double y;
   public PointV2 (double x, double y) { ... }
   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;
   }
}
```

D.T.

```
1  String s = "(2, 3)";
2  PointV1 p1 = new PointV1(2, 3);
3  PointV1 p2 = new PointV1(2, 3);
4  PointV1 p3 = new PointV1(4, 6);
5  System.out.println(p1 == p2);   /* false */
6  System.out.println(p2 == p3);   /* false */
7  System.out.println(p1.equals(p1));   /* true */
8  System.out.println(p1.equals(null));   /* false */
9  System.out.println(p1.equals(s));   /* false */
10 System.out.println(p1.equals(p2));   /* false */
11 System.out.println(p2.equals(p3));   /* false */
```

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



# The equals Method: Overridden Version

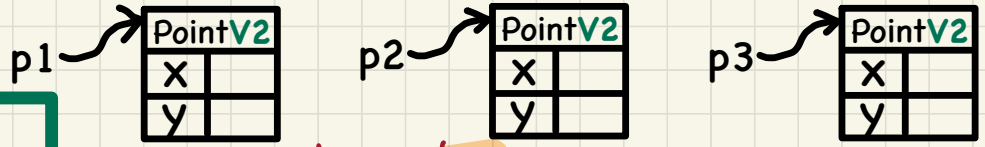
Example 2

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




(A) Two objects are reference-equal.

(B) Two objects are contents-equal.

1- If (A) is true, then (B) is true.

2- If (B) is true, then (A) is true.

¬ x.equals(null)

↳ x.equals(null)

should always return false

return true means x == null

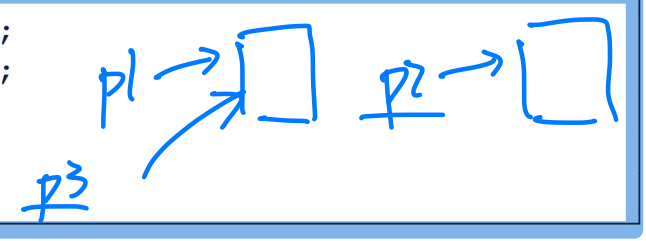
↳ NPE would have occurred.

# assertSame vs. assertEquals

**assertSame**(exp1, exp2)

- Passes if exp1 and exp2 are references to the same object
  - ≈ **assertTrue**(exp1 == exp2)
  - ≈ **assertFalse**(exp1 != exp2)

```
PointV1 p1 = new PointV1(3, 4);
PointV1 p2 = new PointV1(3, 4);
PointV1 p3 = p1;
assertSame(p1, p3);
assertSame(p2, p3);
```




**assertEquals**(exp1, exp2)

- ≈ exp1 == exp2 if exp1 and exp2 are **primitive** type

```
int i = 10;
int j = 20;
assertEquals(i, j);
```

# assertEquals: Reference Comparison or Not

**assertEquals**(exp1, exp2) → ref. types

- ≈ 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) → default version of equals
p2.eq.(p3) → default version of equals

**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);
assertEquals(p2, p3);
assertEquals(p3, p2);
```

D.T.
(p3).eq.(p2) → invoke the overridden version in PointV2

* given that both p2 and p3 store (3,4), is p3.equals(p2) returning true? (F)

(p1 == p2) (F)
"p2 == p3" (F)

ref type ref type

① assertEquals( exp1 , exp2 )

context object argument

↳ assertTrue( exp1.equals(exp2) )

D.T. determines version of equals to invoke.

② assertEquals( exp2 , exp1 )

↳ assertTrue( exp2.equals(exp1) )

Commutativity

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

In Java:

$$p \;\&\&\; q \;\overset{?}{\equiv}\; q \;\&\&\; p$$

No!!!

# 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");
    }
}
```

p && q

① Evaluate p.

② 1 If p eval. to true then eval. q.

② 2 If p eval. to false
↳ skip eval. of q.

# 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");
}
```

p || q

① Evaluate p

② If p eval. to false, then eval. q.

② If p eval. to true, then skip eval. of q.