

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;  
    }  
}
```

PointV2	
x	
y	

The equals Method: Overridden Version

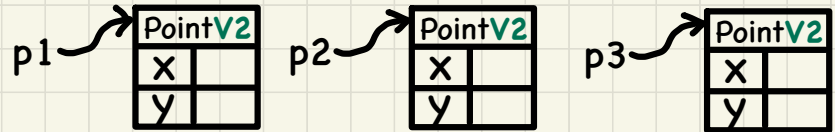
Example 1: Trace L10

```
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

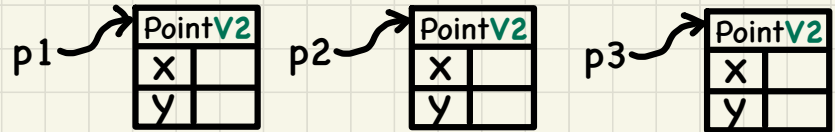
Example 1: Trace L11

```
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)); /* [ ] */
```



The equals Method:

To Override or Not?

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

extends

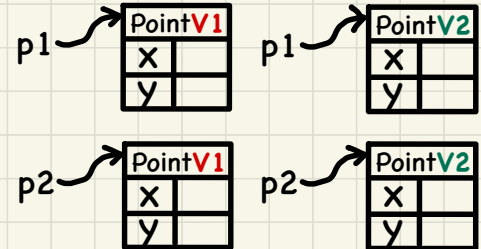
extends

```
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;  
    }  
}
```

```
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

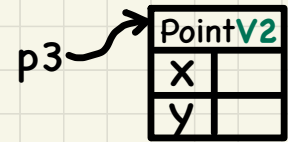
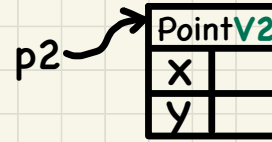
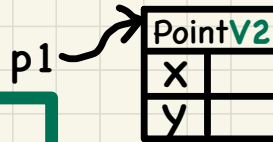
Example 2

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



(A) Two objects are **reference**-equal.

(B) Two objects are **contents**-equal.

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

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)

- $\approx$ `exp1.equals(exp2)` if exp1 and exp2 are **reference** type

Case 1: If `equals` is **not** explicitly overridden in exp1's declared type
 $\approx$ **assertSame**(exp1, exp2)

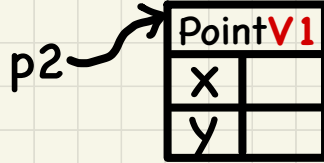
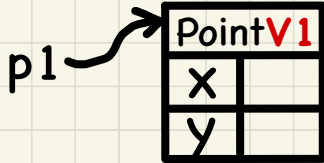
```
PointV1 p1 = new PointV1(3, 4);  
PointV1 p2 = new PointV1(3, 4);  
PointV2 p3 = new PointV2(3, 4);  
assertEquals(p1, p2);  
assertEquals(p2, p3);
```

Case 2: If `equals` is explicitly **overridden** in exp1's declared type
 $\approx$ `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);
```

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;
    }
}
```

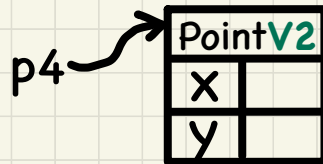
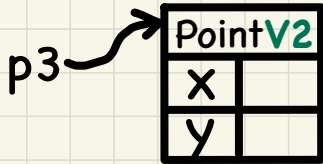
extends

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



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

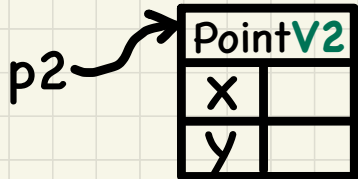
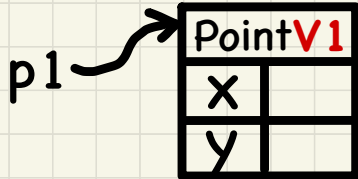
extends

```
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 }  
        Point other = (PointV2) obj;  
        return this.x == other.x  
            && this.y == other.y;  
    }  
}
```

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));
}
```

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



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: 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 }
```

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 }
```

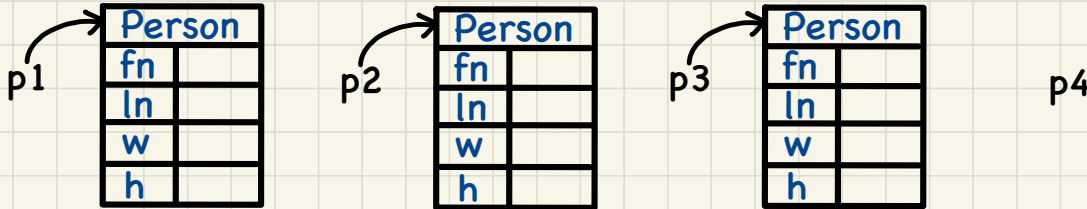
Q: At Line 9 of **PersonCollector**'s **equals** method which version of the **equals** method is called?

```
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 }
```

Testing Equality of Person/PersonCollector in JUnit (1)

@Test

```
public void testPersonCollector() {  
    Person p1 = new Person("A", "a", 180, 1.8);  
    Person p2 = new Person("A", "a", 180, 1.8);  
    Person p3 = new Person("B", "b", 200, 2.1);  
    Person p4 = p3;  
    assertFalse(p1 == p2); assertTrue(p1.equals(p2));  
    assertTrue(p3 == p4); assertTrue(p3.equals(p4));  
}
```

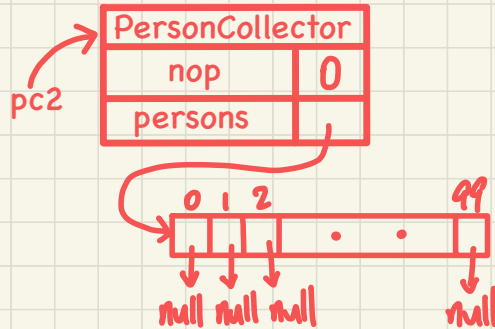
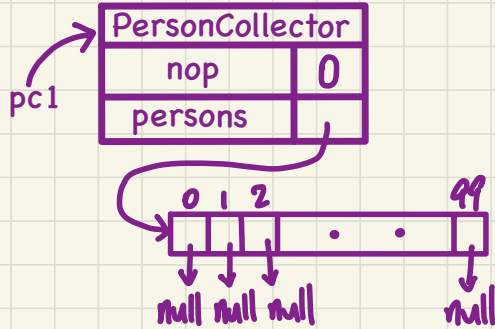


```
public class Person {  
    private String firstName; private String lastName;  
    private double weight; private double height;  
    public boolean equals(Object obj) {  
        if(this == obj) { return true; }  
        if(obj == null || this.getClass() != obj.getClass()) { return false; }  
        Person other = (Person) obj;  
        return  
            this.weight == other.weight  
            && this.height == other.height  
            && this.firstName.equals(other.firstName)  
            && this.lastName.equals(other.lastName);  
    }  
}
```

Testing Equality of **Person/PersonCollector** in **JUnit** (2)

(continued from **testPersonCollector**)

```
PersonCollector pc1 = new PersonCollector();  
PersonCollector pc2 = new PersonCollector();  
assertFalse(pc1 == pc2); assertTrue(pc1.equals(pc2));
```



Q: How about **assertTrue**(pc2.equals(pc1))?

```
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() { ... }  
}  
  
public boolean equals(Object obj) {  
    if(this == obj) { return true; }  
    if(obj == null || this.getClass() != obj.getClass()) { return false; }  
    PersonCollector other = (PersonCollector) obj;  
    boolean equal = false;  
    if(this.nop == other.nop) {  
        equal = true;  
        for(int i = 0; equal && i < this.nop; i++) {  
            equal = this.persons[i].equals(other.persons[i]);  
        }  
    }  
    return equal;  
}
```

Testing Equality of Person/PersonCollector in JUnit (3)

(continued from [testPersonCollector](#))

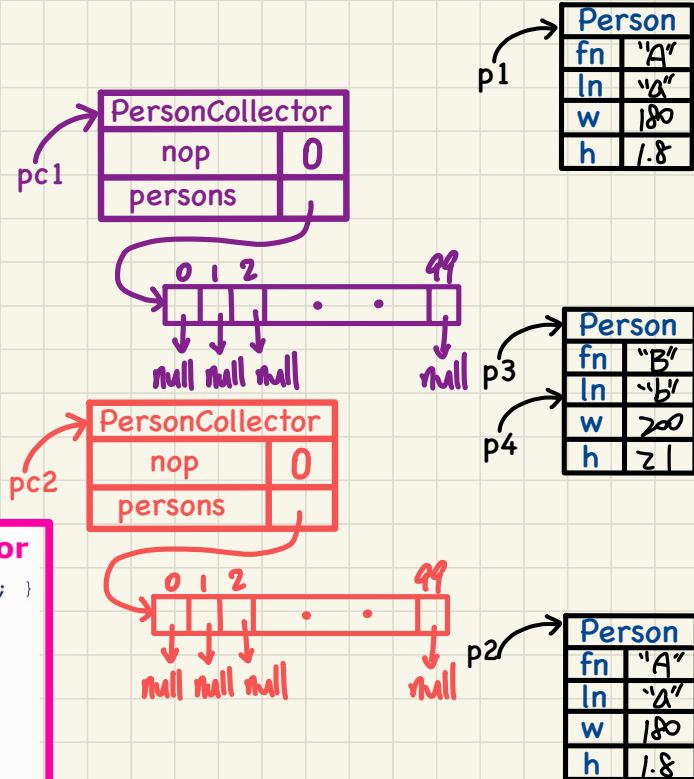
```
pc1.addPerson(p1);
assertFalse(pc1.equals(pc2));
pc2.addPerson(p2);
assertFalse(pc1.getPersons()[0] == pc2.getPersons()[0]);
assertTrue(pc1.getPersons()[0].equals(pc2.getPersons()[0]));
assertTrue(pc1.equals(pc2));
pc1.addPerson(p3);
pc2.addPerson(p4);
assertTrue(pc1.getPersons()[1] == pc2.getPersons()[1]);
assertTrue(pc1.getPersons()[1].equals(pc2.getPersons()[1]));
assertTrue(pc1.equals(pc2));
```

```
public boolean equals(Object obj) {
    if(this == obj) { return true; }
    if(obj == null || this.getClass() != obj.getClass()) { return false; }
    Person other = (Person) obj;
    return
        this.weight == other.weight
        && this.height == other.height
        && this.firstName.equals(other.firstName)
        && this.lastName.equals(other.lastName);
}
```

Person

```
public boolean equals(Object obj) {
    if(this == obj) { return true; }
    if(obj == null || this.getClass() != obj.getClass()) { return false; }
    PersonCollector other = (PersonCollector) obj;
    boolean equal = false;
    if(this.nop == other.nop) {
        equal = true;
        for(int i = 0; equal && i < this.nop; i++) {
            equal = this.persons[i].equals(other.persons[i]);
        }
    }
    return equal;
}
```

PersonCollector



Testing Equality of Person/PersonCollector in JUnit (4)

(continued from [testPersonCollector](#))

```
pc1.addPerson(new Person("A", "a", 175, 1.75));  
pc2.addPerson(new Person("A", "a", 165, 1.55));  
assertFalse(pc1.getPersons()[2] == pc2.getPersons()[2]);  
assertFalse(pc1.getPersons()[2].equals(pc2.getPersons()[2]));  
assertFalse(pc1.equals(pc2));
```

Person	
fn	
ln	
w	
h	

```
public boolean equals(Object obj) {  
    if(this == obj) { return true; }  
    if(obj == null || this.getClass() != obj.getClass()) { return false; }  
    Person other = (Person) obj;  
    return  
        this.weight == other.weight  
        && this.height == other.height  
        && this.firstName.equals(other.firstName)  
        && this.lastName.equals(other.lastName);  
}
```

Person

```
public boolean equals(Object obj) {  
    if(this == obj) { return true; }  
    if(obj == null || this.getClass() != obj.getClass()) { return false; }  
    PersonCollector other = (PersonCollector) obj;  
    boolean equal = false;  
    if(this.nop == other.nop) {  
        equal = true;  
        for(int i = 0; equal && i < this.nop; i++) {  
            equal = this.persons[i].equals(other.persons[i]);  
        }  
    }  
    return equal;  
}
```

PersonCollector

