Lecture 9

Wednesday October 2

The equals Method: To Override or Not?

```
class Object {
   boolean equals(Object obj) {
      retuen this == obj;
   }
}
```

extends

extends

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

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

assertEquals(obj1, obj2)

c.o.

↳ obj1.equals(obj2)

assertEquals(obj2, obj1)

c.o

↳ obj2.equals(obj1)

assertEquals(exp1, exp2)

- ≈ `exp1.equals(exp2)` if exp1 and exp2 are **reference** type

Case 1: If `equals` is not explicitly overridden in *obj*1's declared type

≈ **assertSame**(obj1, obj2)

```
PointV1 p1 = new PointV1(3, 4);
PointV1 p2 = new PointV1(3, 4);
PointV2 p3 = new PointV2(3, 4);
assertEquals(p1, p2);   × /* ∵ different PointV1 objects */
assertEquals(p2, p3);   × /* ∵ different types of objects */
```

Case 2: If `equals` is explicitly overridden in *obj*1's declared type

≈ `obj1.equals(obj2)`

```
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);   × /* ≈
```

equals (Object obj)

PointV1

PointV1

PointV2

p1 = new..

p2 = new...

p3 = new....

at runtime

p2 == p3

① p1 == p2 ✓ ∵ same type

② p1.equals(p2) ref type

PointV1 which is an object

③ p2 == p3 X not compile ∵ diff. types.

④ p2.equals(p3) p2 is a ref. type.

every ref type is an object

Testing Default Equality of Points in JUnit

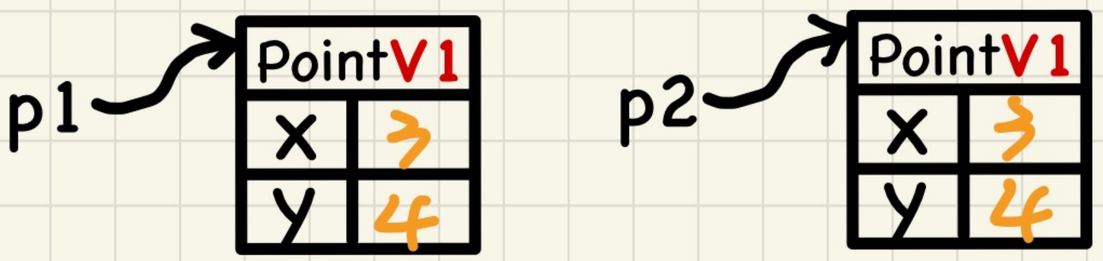
```
@Test
public void testEqualityOfPointV1() {
  PointV1 p1 = new PointV1(3, 4); PointV1 p2 = new PointV1(3, 4);
  assertFalse(p1 == p2); assertFalse(p2 == p1);
  /* assertSame(p1, p2); assertSame(p2, p1); */ /* both fail */
  assertFalse(p1.equals(p2)); assertFalse(p2.equals(p1));
  assertTrue(p1.x == p2.x && p2.y == p2.y);
}
```

default

p1 == p2

default

p2 == p1




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

extends

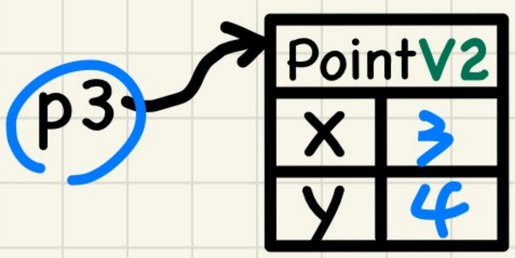
```
class PointV1 {
  double x;
  double y;
  PointV1 (double x, double y) {
    this.x = x;
    this.y = y;
  }
}
```

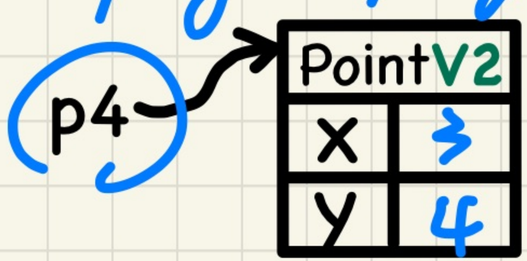
Testing Overridden Equality of Points in JUnit

```
@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);
}
```

overridden

p3.x == p4.x &&
p3.y == p4.y






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

extends

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

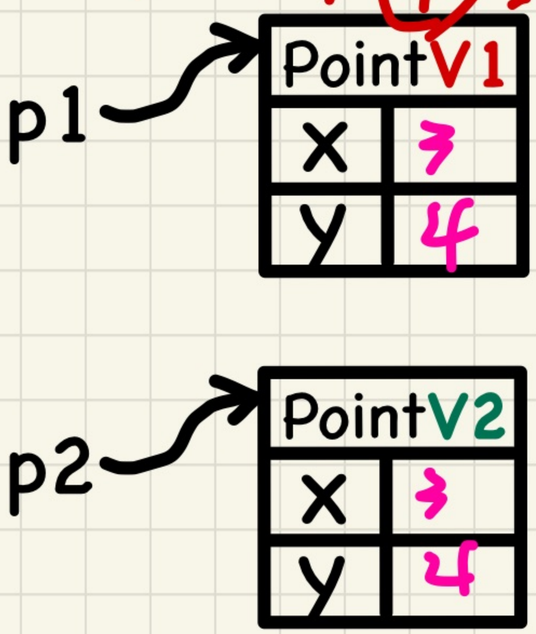
Testing Equality of Points in JUnit

```
@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.equals(p2) → default → p1 == p2

p2.equals(p1) → Overridden

p1 == p2 ? compile (F) — not compiling ∵ p1 and p2 different types




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

extends

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

extends

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

p2 p1 ?

```
Point V1   p1 = new Point V1 (3, 4);
Point V2   p2 = new Point V2 (3, 4);
```

Known:

p1.equals(p2) → F (different addresses)

p2.equals(p1) → F (different types)

boolean CompareV1V2(PointV1 p1, PointV2 p2){

return p1.x == p2.x
&& p1.y == p2.y;

}

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

```
if ( obj == null || this.getC != obj.getC){
    return false ;
}
```

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

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

Q1: At Lines 10 and 11 which version of the **equals** method is called?

Q2: At Line 5, will there be a **NullPointerException** if obj == **null**?

Q3: At Line 5, what if we change it to:

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

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

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

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

Person

Pt

Pt

Q: At Line 10 of PersonCollector which version of the equals method is called?

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

(10)

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

this.persons[i].weight ==

other.persons[i].weight

&&

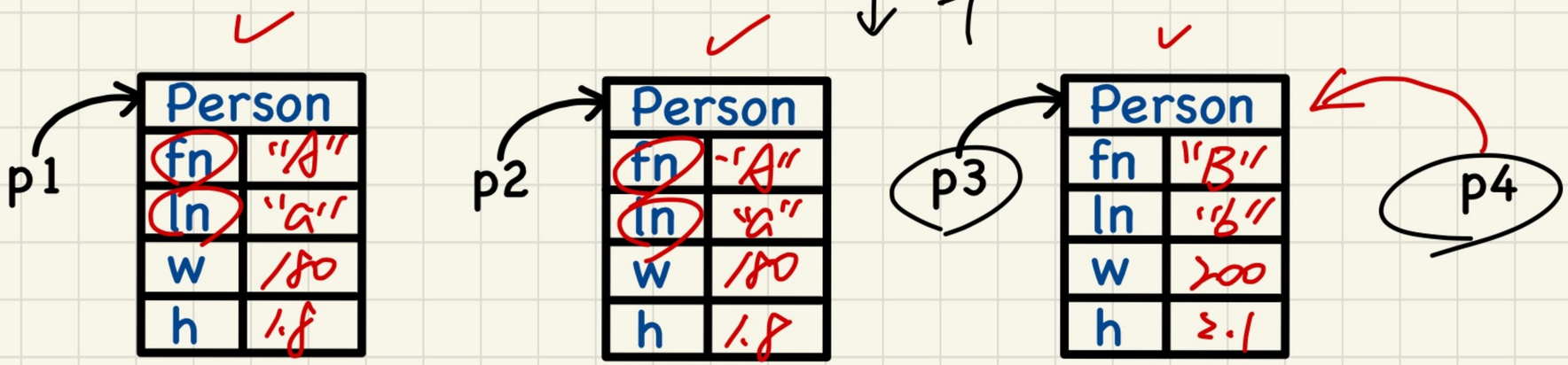
.

.

.

Testing Equality of Person/PersonCollector in JUnit (1)

```java
@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));
```

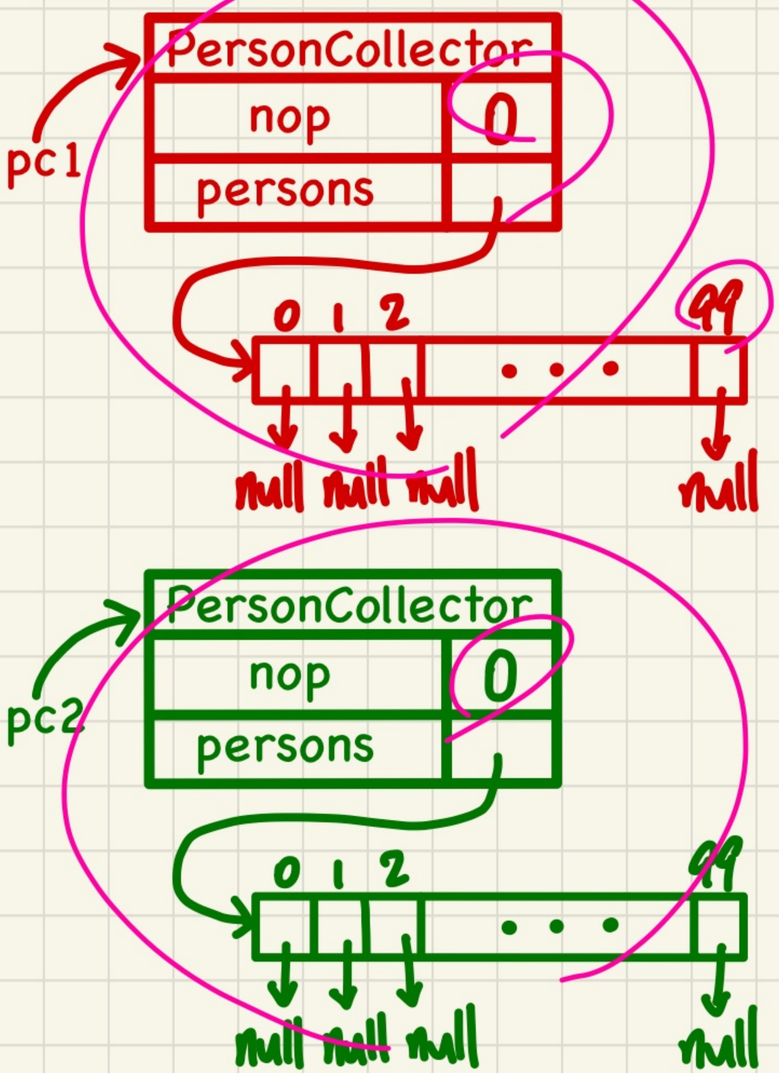



```java
class Person {
  String firstName; String lastName; double weight; double height;
  boolean equals(Object obj) {
    if(this == obj) { return true; }
    if(obj == null || this.getClass() != obj.getClass()) {
      return false; }
    Person other = (Person) obj;
    return
        this.weight == 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 {
  Person[] persons; int nop; /* number of persons */
  public PersonCollector() { ... }
  public void addPerson(Person p) { ... }
}
```

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

Testing Equality of Person/PersonCollector in JUnit (3)

```
pc1.addPerson(p1);
assertFalse(pc1.equals(pc2));

pc2.addPerson(p2);
assertFalse(pc1.persons[0] == pc2.persons[0]);
assertTrue(pc1.persons[0].equals(pc2.persons[0]));
assertTrue(pc1.equals(pc2));

pc1.addPerson(p3); pc2.addPerson(p4);
assertTrue(pc1.persons[1] == pc2.persons[1]);
assertTrue(pc1.persons[1].equals(pc2.persons[1]));
assertTrue(pc1.equals(pc2));
```

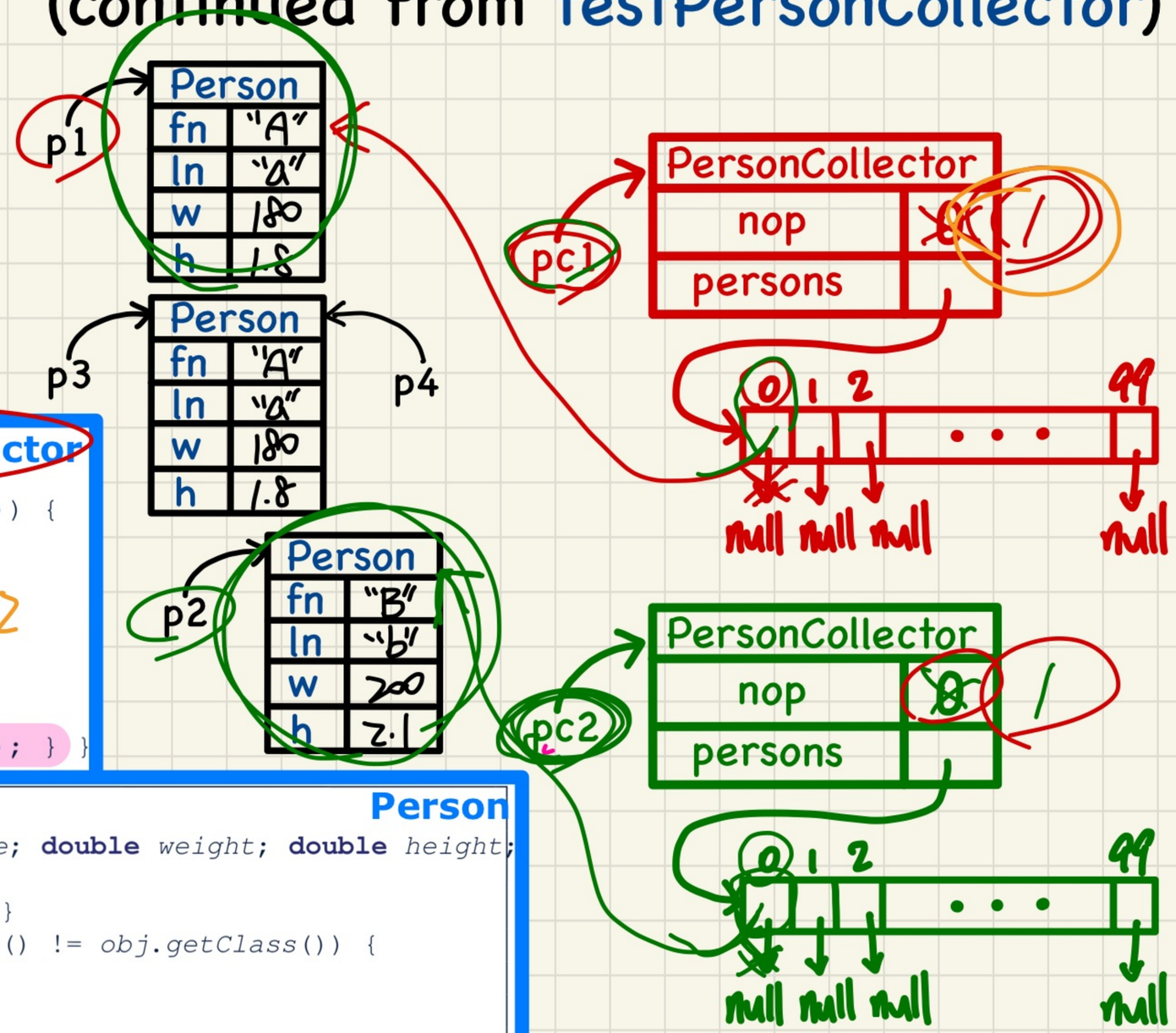
PersonCollector

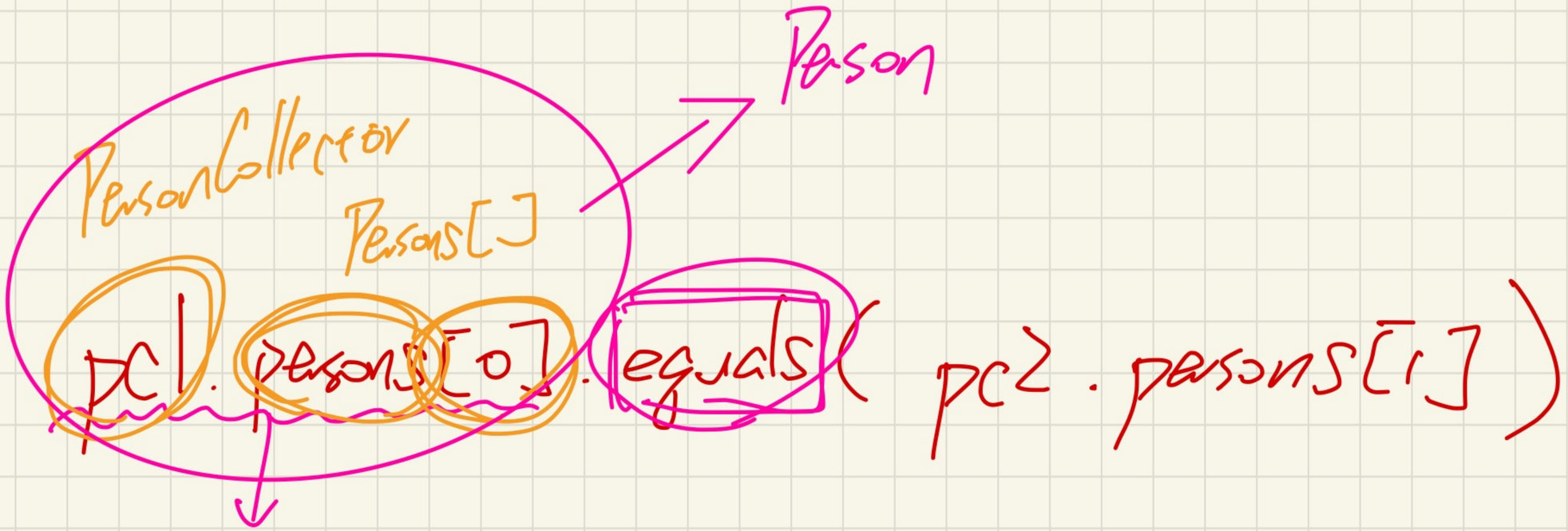
```
1  boolean equals(Object obj) {
2    if(this == obj) { return true; }
3    if(obj == null || this.getClass() != obj.getClass()) {
4      return false; }
5    PersonCollector other = (PersonCollector) obj;
6    boolean equal = false;
7    if(this.nop == other.nop) {
8      equal = true;
9      for(int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

Person

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

(continued from testPersonCollector)





C.O -

```
@Override
boolean equals(. - )
```

Ordering between Employees

name	id	salary
alan	2	4500.31
mark	3	3450.67
tom	1	3450.67

attributes

tom.id < alan.id < mark.id

mark.salary = mark.salary < alan.salary

tom

Sorting: from smallest to largest

Sorting based on id's

(smaller id's come first)

tom < alan < mark

Sorting based on salaries and id's

(higher salaries and smaller id's come first)

alan tom < mark