Lecture 14 - Oct. 29

Call by Value

equals: PersonsCollector
Call by Value

Announcements/Reminders

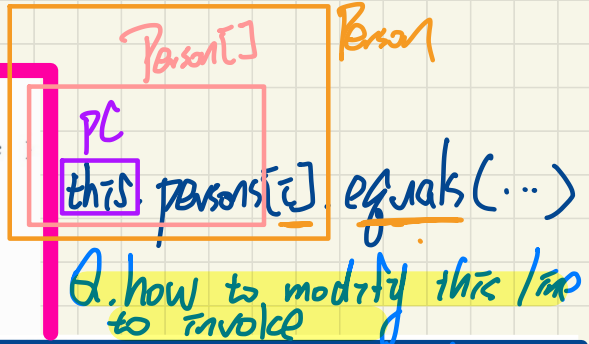
- ProgTest1 marks, submissions, grading tests released
- ProgTest2 tomorrow during your enrolled lab session
- Lab3 due this Friday at noon

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?



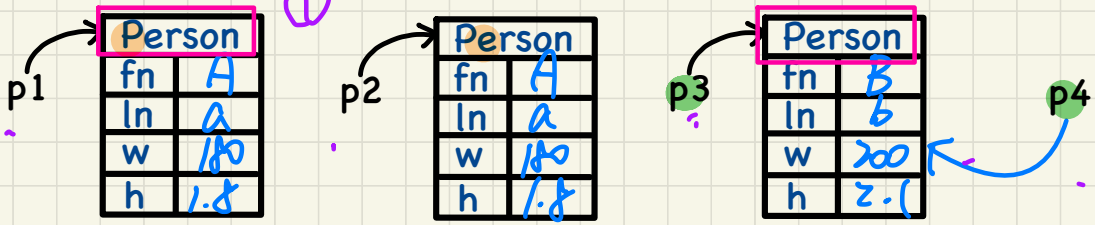

```
1  public class Person {
2    private String firstName; private String lastName;
3    private double weight; private double height;
4    public boolean equals(Object obj) {
5      if(this == obj) { return true; }
6      if(obj == null || this.getClass() != obj.getClass()) { return false; }
7    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

```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));
```

* p1.equals(p2) (T) — which parts?

** p3.equals(p4) (T)

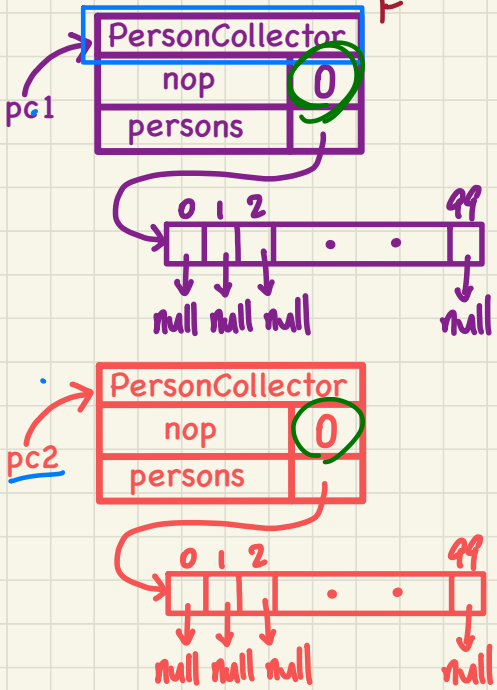



```java
public class Person {
  private String firstName; private String lastName;
  private double weight; private double height;
  public boolean equals(Object obj) {
    if(this == obj) { return true; }
    if(obj == null || this.getClass() != obj.getClass()) { return false; }
    Person other = (Person)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

F



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;
}
```

T T 0 < 0 F → not even the 1st it.

Fail.

Testing Equality of Person/PersonCollector in JUnit (3)

(continued from testPersonCollector)

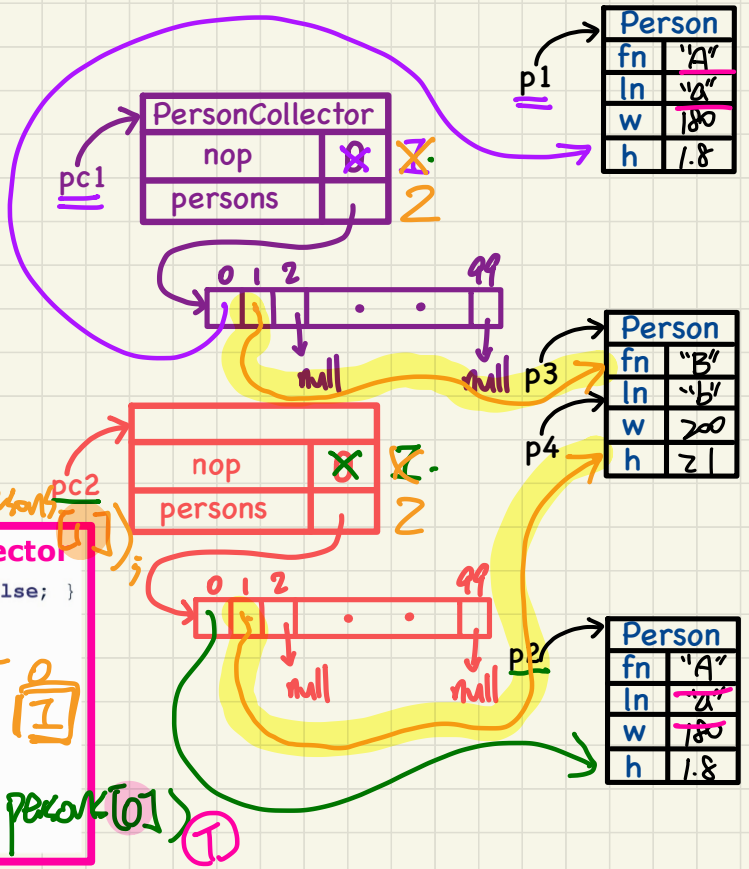
```
pc1.addPerson(p1);
assertFalse(pc1.equals(pc2));
pc2.addPerson(p2);
assertFalse(pc1.getPersons()[0] == pc2.getPersons()[0]);
assertTrue(pc1.getPersons()[0].equals(pc2.getPersons()[0]));
assertTrue(pc1.equals(pc2));
pc1.addPerson(p3);
pc2.addPerson(p4);
assertTrue(pc1.getPersons()[1] == pc2.getPersons()[1]);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

Person

```
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);
}
```

PersonCollector

```
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 (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));
```

* 3rd iteration:

equals(pc1.persons[2].equals(pc2.persons[2])

Person
fn A
ln a
w 175
h 1.75

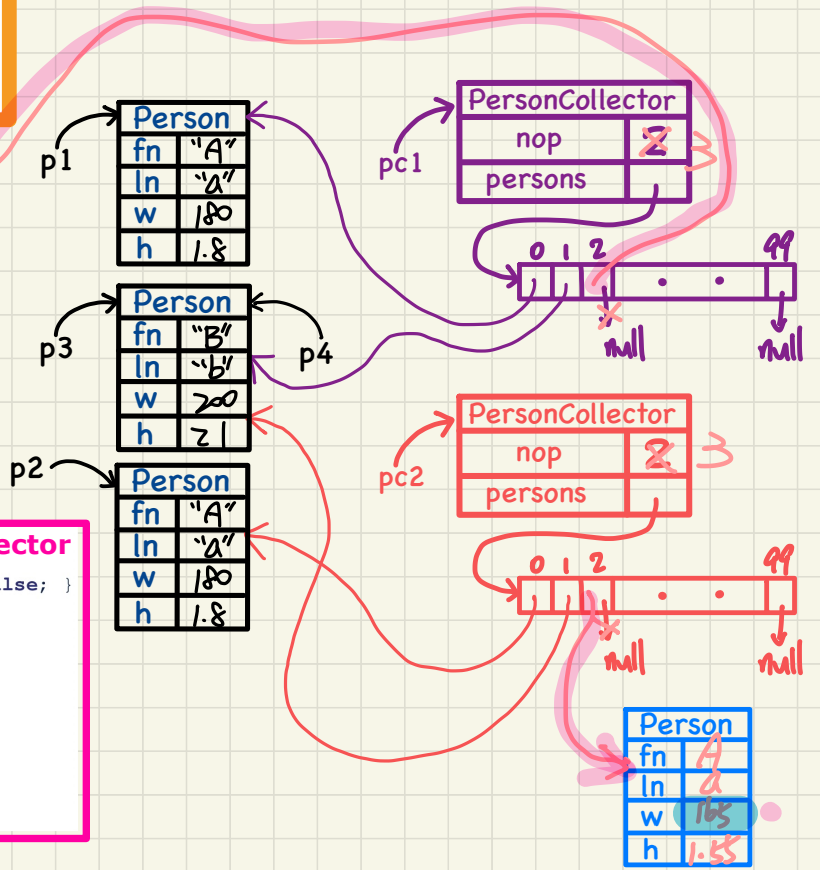
```
public boolean equals(Object obj) {
  if(this == obj) { return true; }
  if(obj == null || this.getClass() != obj.getClass()) { return false; }
  Person other = (Person) obj;
  return
       this.weight == other.weight
    && this.height == other.height
    && this.firstName.equals(other.firstName)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

Person

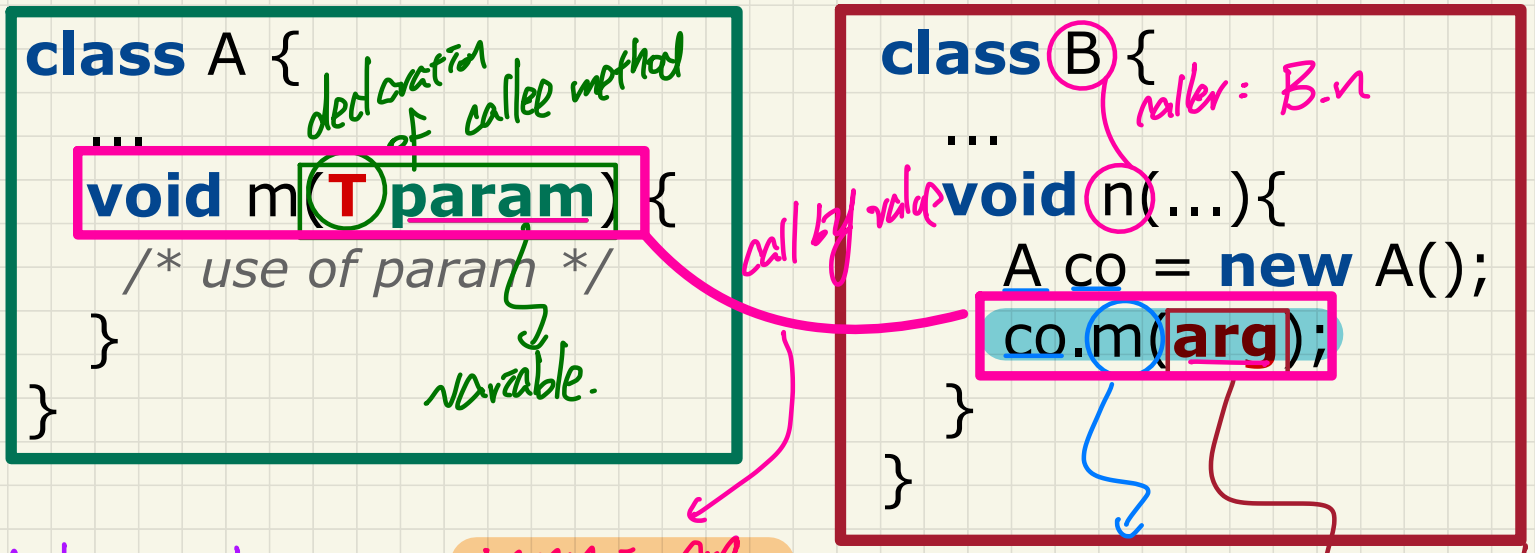
```
public boolean equals(Object obj) {
  if(this == obj) { return true; }
  if(obj == null || this.getClass() != obj.getClass()) { return false; }
  PersonCollector other = (PersonCollector) obj;
  boolean equal = false;
  if(this.nop == other.nop)
    equal = true;
  for(int i = 0; equal && i < this.nop;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

PersonCollector

i: 0, 1, 2*



Method Call: Callee vs. Caller



Call by value:

when making a method call, there's an implicit variable assignment:

param = arg;

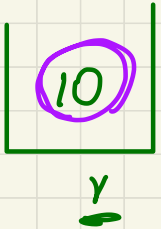
Call by Value: Primitive Argument

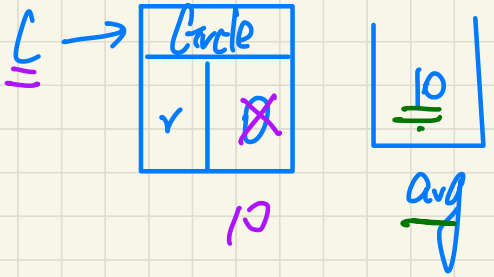
```
class Circle {
   int radius;
   void setRadius(int r) {
      this.radius = r;
   }
}
```

Call by value:

r = arg

```
class CircleUser {
   ...
   Circle c = new Circle();
   int arg = 10;
   c.setRadius(arg);
   }
}
```



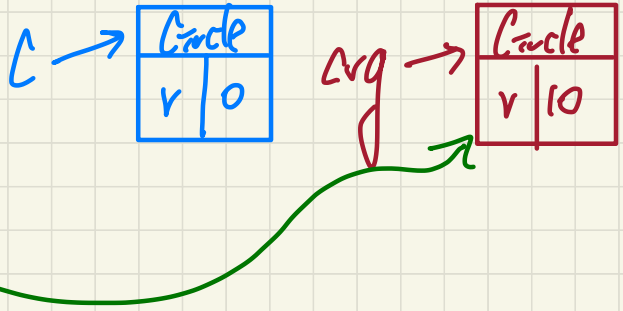


Call by Value: Reference Argument

```
class Circle {
   int radius;
   Circle() {}
   Circle(int r) {
      this.radius = r;
   }
   void setRadius(Circle c) {
      this.radius = c.radius;
   }
}
```

```
class CircleUser {
   ...
   Circle c = new Circle();
   Circle arg = new Circle(10);
   c.setRadius(arg);
   }
}
```

Call by value: c = arg

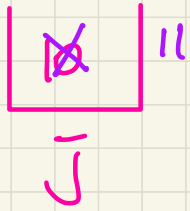


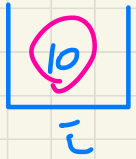
Call by Value: Re-Assigning Primitive Parameter

```
public class Util {
  void reassignInt(int j) {
    j = j + 1; }
  void reassignRef(Point q) {
    Point np = new Point(6, 8);
    q = np; }
  void changeViaRef(Point q) {
    q.moveHorizontally(3);
    q.moveVertically(4); } }
```

j = i

```
1  @Test
2  public void testCallByVal() {
3    Util u = new Util();
4    int i = 10;
5    assertTrue(i == 10);
6    u.reassignInt(i);
7    assertTrue(i == 10);
8  }
```





i is untouched by the method.

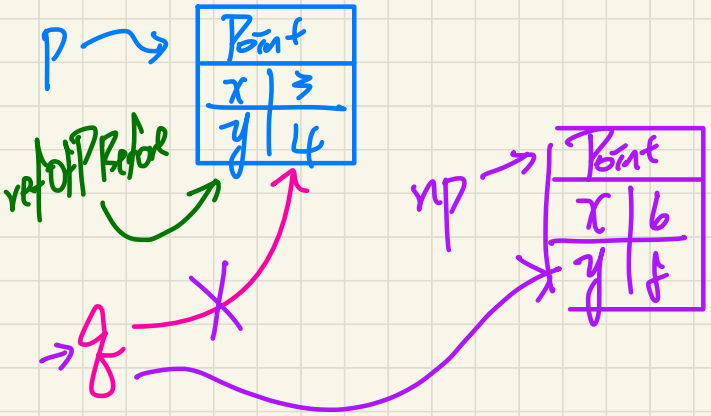
Call by Value: Re-Assigning Reference Parameter

```
public class Util {
  void reassignInt(int j) {
    j = j + 1; }
  void reassignRef(Point q) {
    Point np = new Point(6, 8);
    q = np; }
  void changeViaRef(Point q) {
    q.moveHorizontally(3);
    q.moveVertically(4); } }
```

```
1  @Test
2  public void testCallByRef_1() {
3    Util u = new Util();
4    Point p = new Point(3, 4);
5    Point refOfPBefore = p;
6    u.reassignRef(p);
7    assertTrue(p == refOfPBefore);
8    assertTrue(p.getX() == 3);
9    assertTrue(p.getY() == 4);
10 }
```

Call by value: q = p

address of the value of p stays the same.

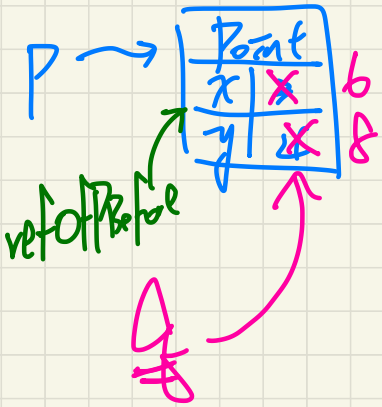



```
public class Point {
  private int x;
  private int y;
  public Point(int x, int y) {
    this.x = x;
    this.y = y;
  }
  public int getX() { return this.x; }
  public int getY() { return this.y; }
  public void moveVertically(int y){ this.y += y; }
  public void moveHorizontally(int x){ this.x += x; }
}
```

Call by Value: Calling Mutator on Reference Parameter

```
public class Util {
  void reassignInt(int j) {
    j = j + 1; }
  void reassignRef(Point q) {
    Point np = new Point(6, 8);
    q = np; }
  void changeViaRef(Point q) {
    q.moveHorizontally(3);
    q.moveVertically(4); } }
```

```
1  @Test
2  public void testCallByRef_2() {
3    Util u = new Util();
4    Point p = new Point(3, 4);
5    Point refOfPBefore = p;
6    u.changeViaRef(p);
7    assertTrue(p == refOfPBefore);
8    assertTrue(p.getX() == 6);
9    assertTrue(p.getY() == 8);
10 }
```



call by value:

q = p

```
public class Point {
  private int x;
  private int y;
  public Point(int x, int y) {
    this.x = x;
    this.y = y;
  }
  public int getX() { return this.x; }
  public int getY() { return this.y; }
  public void moveVertically(int y){ this.y += y; }
  public void moveHorizontally(int x){ this.x += x; }
}
```

Call by Value: Invoking the Overridden equals

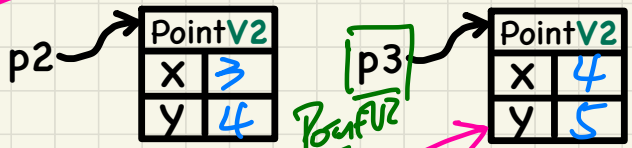
```
public class Object {
   ...
   public boolean equals(Object obj) {
      return this == obj;
   }
}
```

extends

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

Call by value: obj = p3

```
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 ST.

obj

ST: Object

obj.x

obj.y