

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() { ... }  
}
```

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

```
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

```
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("Ä", "ä", 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

Person	
fn	A
ln	a
w	180
h	1.8

p2

Person	
fn	A
ln	a
w	180
h	1.8

p3

Person	
fn	B
ln	b
w	200
h	2.1

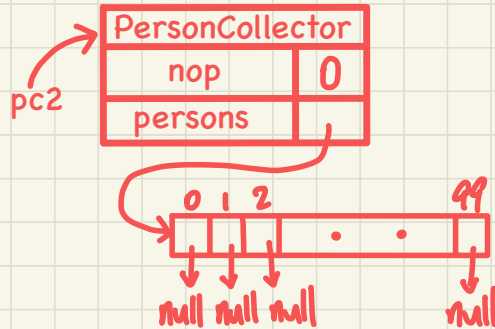
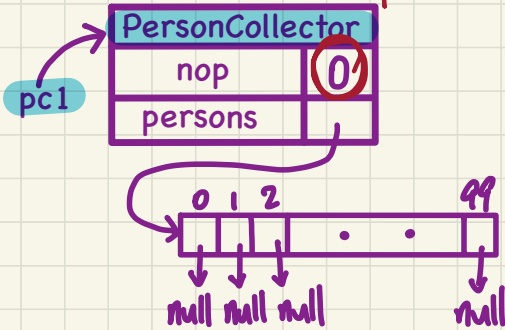
p4

```
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;  
    }  
}
```

Handwritten notes and annotations:

- Annotations on the `equals` method: `if(this == obj)` is marked with a blue 'X' and `return true;` is marked with a blue 'X'.
- Annotations on the `if(obj == null || this.getClass() != obj.getClass())` line: `return false;` is marked with a blue 'X'.
- Annotations on the `if(this.nop == other.nop)` line: `equal = true;` is marked with a blue checkmark.
- Annotations on the `for` loop: `for(int i = 0; equal && i < this.nop; i++)` is highlighted with a blue box. Handwritten notes include: `0 pc1.nop 0 < 0 → F` and `∴ i < this.nop eval. to false right away → no if. run.`

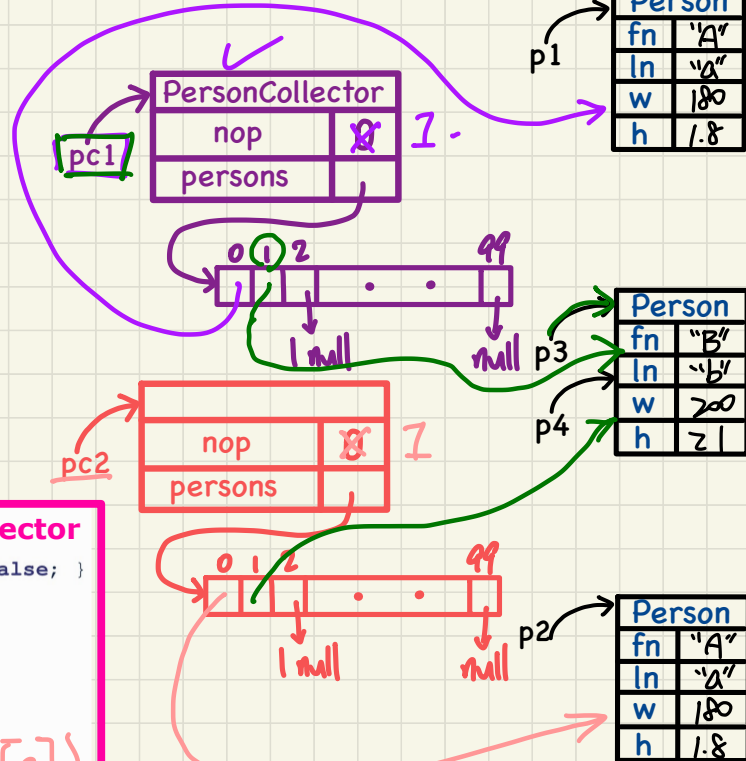
Testing Equality of Person/PersonCollector in JUnit (3)

(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);  
}
```

```
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));
```

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);  
}
```

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;  
    }  
}
```

p1

Person	
fn	"A"
ln	"a"
w	180
h	1.8

p3

Person	
fn	"B"
ln	"b"
w	200
h	2.1

p2

Person	
fn	"A"
ln	"a"
w	180
h	1.8

pc1

PersonCollector	
nop	3
persons	[0, 1, 2, ..., 99]

pc2

PersonCollector	
nop	3
persons	[0, 1, 2, ..., 99]

Person	
fn	A
ln	a
w	165
h	1.55

0
1
2

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

Method Call: Callee vs. Caller

callee

```
class A {  
  ...  
  void m(T param) {  
    /* use of param */  
  }  
}
```

variable (parameter)

```
class B {  
  ...  
  void n(...){  
    A co = new A();  
    co.m(arg);  
  }  
}
```

caller: B.n

A.m callee

param is a copy of arg

concrete value (argument)

Call by Value: Primitive Argument

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

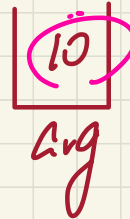
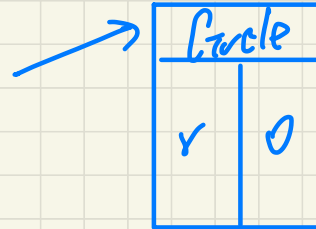
primitive param.

r is a copy of arg

as if:
 $r = arg$

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

primitive argument.



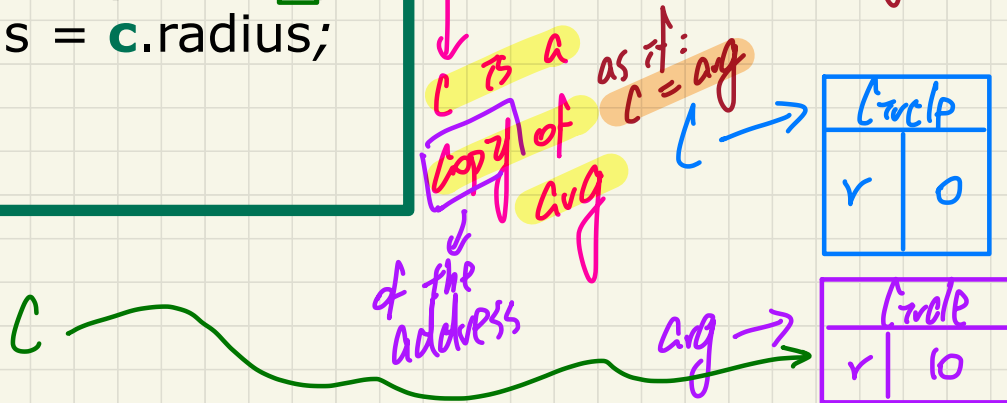
Call by Value: Reference Argument

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

*ref. type
parameter*

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

*ref. type
argument*

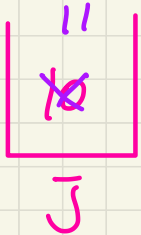


Call by Value: Re-Assigning Primitive Parameter

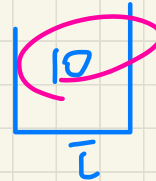
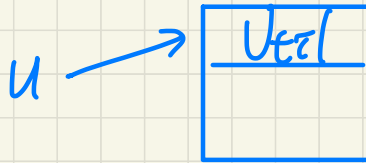
```
public class Util {  
    void reassignInt(int j) {  
        j = j + 1; }  
    void reassignRef(Point q) {  
        Point np = new Point(6, 8);  
        q = np; }  
    void changeViaRef(Point q) {  
        q.moveHorizontally(3);  
        q.moveVertically(4); } }  
}
```

```
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 }
```

arg stays untouched.



call by value:
 $j = i$



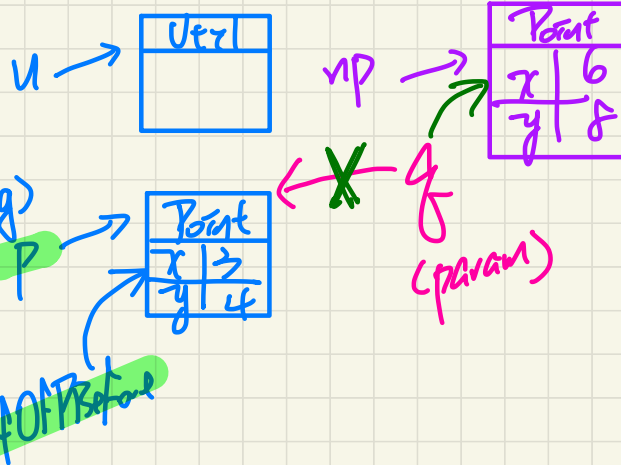
Call by Value: Re-Assigning Reference Parameter

```
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); } }
```

call by value:
✓ $pq = P$

```
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 }
```

never possible
to re-assign
arg. P
using
a method call.

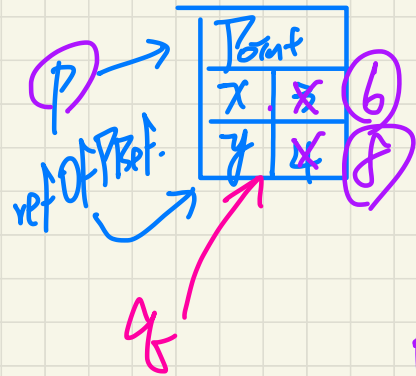


```
public class Point {  
    private int x;  
    private int y;  
    public Point(int x, int y) {  
        this.x = x;  
        this.y = y;  
    }  
    public int getX() { return this.x; }  
    public int getY() { return this.y; }  
    public void moveVertically(int y) { this.y += y; }  
 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



Contents of the object pointed by `p` were modified

```
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; }  
}
```

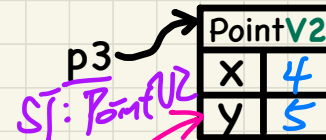
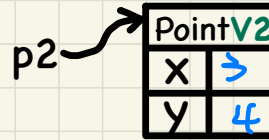
Call by Value: Invoking the Overridden equals

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

extends

```
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); /* 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 */
```



obj
↓
ST: Object

obj = p3 ✓