Lecture 12 - Oct 21

Object Equality

Call by Value: Prim. vs. Ref. Types
Equality: == vs. equals
Default Version of equals in Object Class

Announcements/Reminders

- Today's class: notes template posted
- ProgTest0, ProgTest1 grading process:
  + TAs are due to return results to me Tuesday afternoon.
  + I will take one day or two to finalize the results.
- WrittenTest1 next Wednesday (October 29)
  + Some practice questions on eClass
  + Lecture 12 and Lecture 13 this week **are** covered.
  + Zoom Review Session: 4 PM on Saturday, October 25
- Lab3 to be released

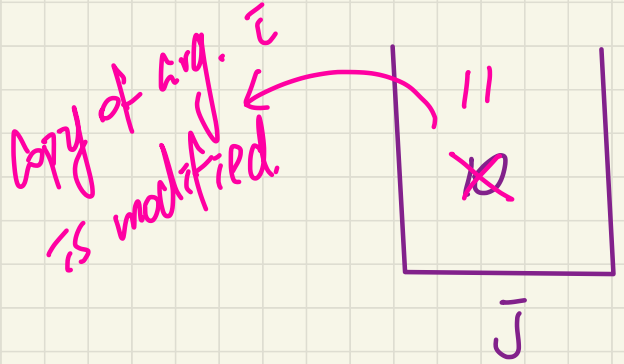
Call by Value: Re-Assigning Primitive Parameter

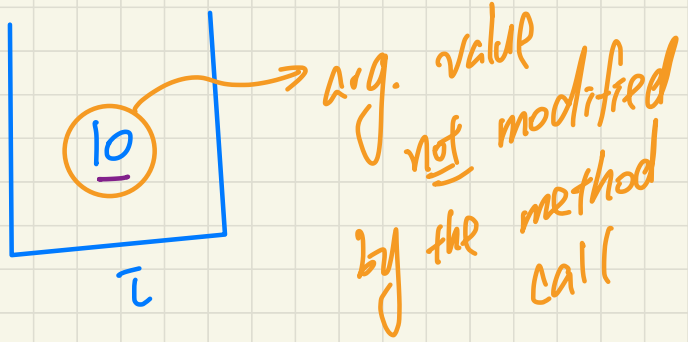
```
public class Util {
  void reassignInt(int j) {
  j = j + 1; }
  void reassignRef(Point q) {
    Point np = new Point(6, 8);
    q = np; }
  void changeViaRef(Point q) {
    q.moveHorizontally(3);
    q.moveVertically(4); } }
```

c.b.v.

5 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  }
```





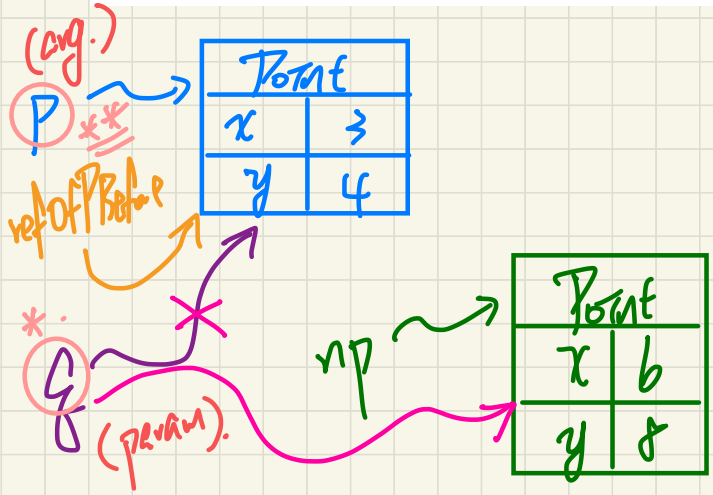
Call by Value: Re-Assigning Reference Parameter

* param. (q) is re-assigned to np.

** arg (p) stays the same.

```
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 }
```

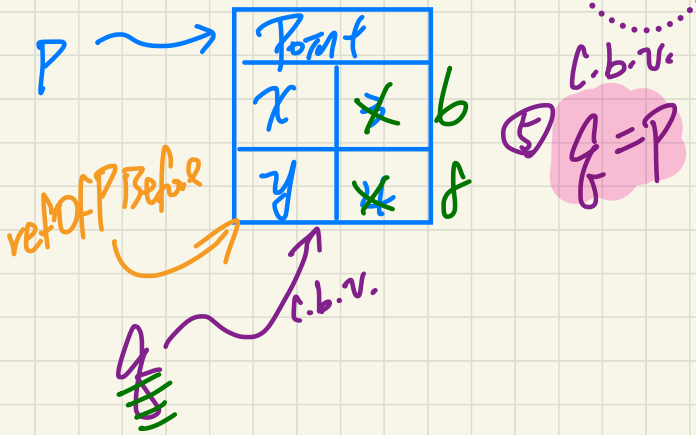



```
public class Point {
  private int x;
  private int y;
  public Point(int x, int y) {
    this.x = x;
    this.y = y;
  }
  public int getX() { return this.x; }
  public int getY() { return this.y; }
  public void moveVertically(int y){ this.y += y; }
  public void moveHorizontally(int x){ this.x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




```
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; }
}
```

Equality

==

equals

Primitive

1. == (value)
2. equals X

```
int i = 23;
int j = 46;
```

i == j → false

Reference

1. == (address/ref)
2. equals
   - ↳ default version?
   - customized version?

```
Point p1 = new Point(...);
Point p2 = new Point(...);
```

p1 == p2 → false ∵ diff. addresses

p1.equals(p2)

↳ Compares result depends on how the Point class defines equals

```
Integer i1 = new I.(23);
Integer i2 = new I.(23);
i1 == i2 → false ∵ diff. refs.
i1.equals(i2) → true
```

i1 → I. | v: 23

i2 → I. | v: 23

tmp

invalid C.O.

i.equals(j) X

↳ compilation error

Object Equality: First Example

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

Q1:

Does PointTester compile, given that **equals** is **not** declared in PointV1?

YES.

Q2:

Where does **equals** come from?

↳ Object

```
class PointTester {
  ... main(...) {
    PointV1 p1 = new PointV1(3, 4);
    PointV1 p2 = new PointV1(3, 4);
    System.out.println(p1.equals(p2));
  }
}
```

boil down into p1 == p2

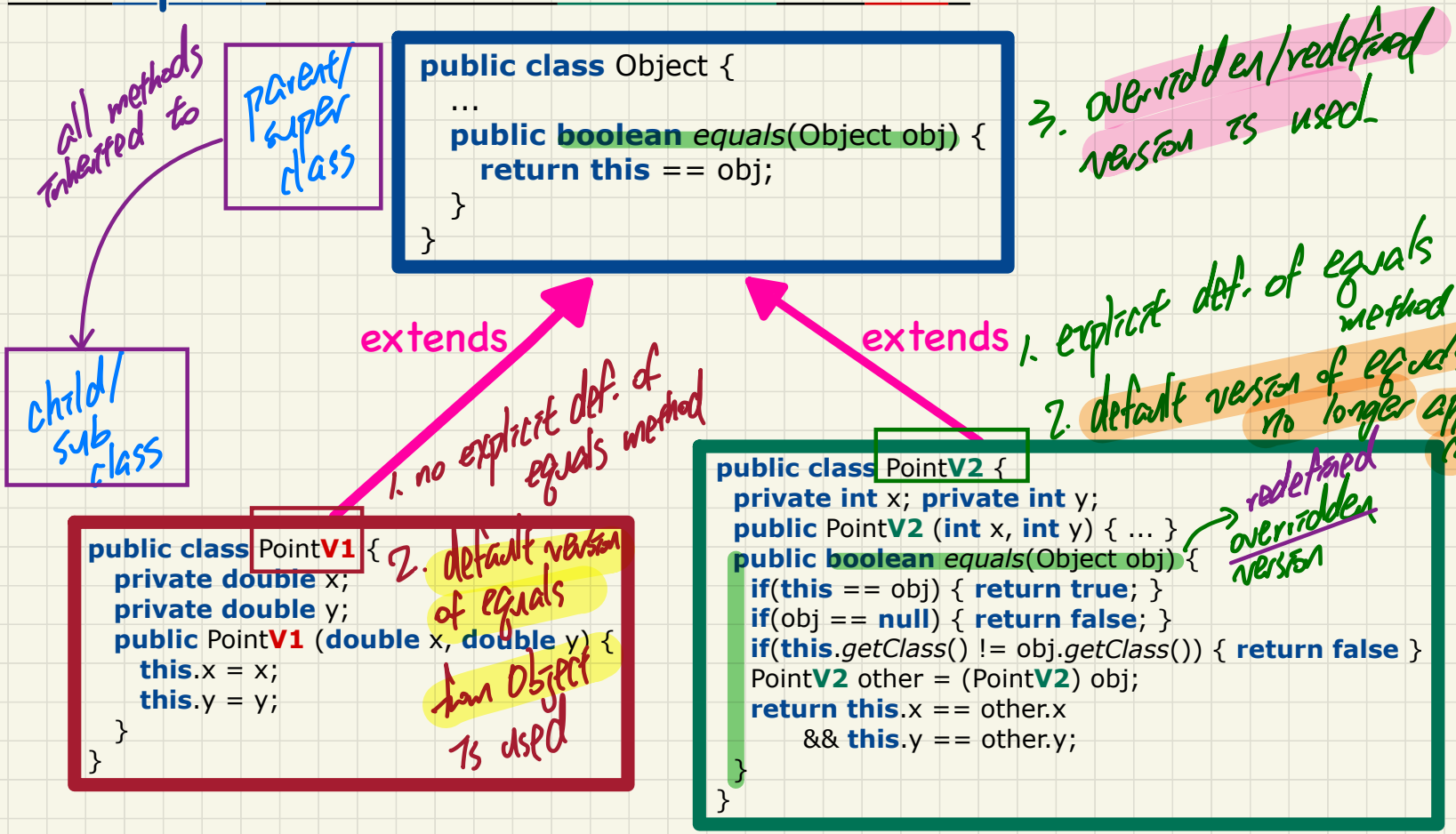
this

1. not explicitly defined in PointV1
2. default version from Object is called.*

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

p1 p2 p2

The equals Method: To Override or Not?



The equals Method: Default Version

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

default version inherited to all classes.

extends

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

Since PointV1 does not explicitly override/redefine equals → default version inherited from Object is used.

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