Lecture 11 - Oct. 19

Object Equality

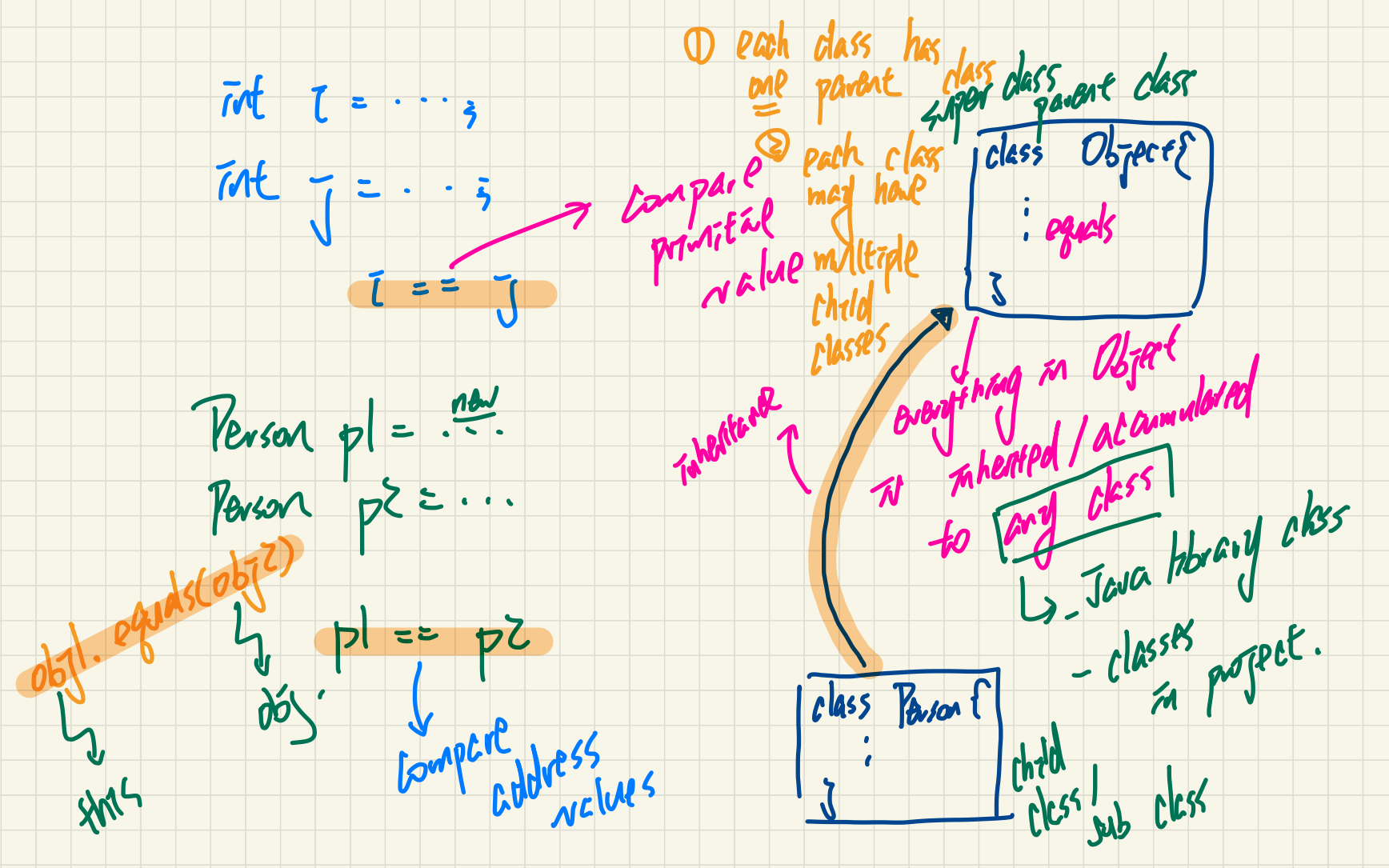
To Override or Not to Override
Overriding equals: 4 Phases

Announcements

- Lab2 due this Friday
- Look ahead: WrittenTest2 & ProgTest2
    + **Important Exercise**:

      Use debugger to explore execution paths

      in the console testers & JUnit tests


int i = ...;
int j = ...;
i == j
compare primitive value
① each class has one parent class
super class
parent class
② each class may have multiple child classes
class Object{
equals
}
inheritance
everything in Object inherited / accumulated to any class
in
Java library class
- classes in project.
class Person{
}
child class / sub class
Person p1 = new ...
Person p2 = ...
obj1.equals(obj2)
this
obj
p1 == p2
compare address values

$x == y$

① x, y are primitive vars.

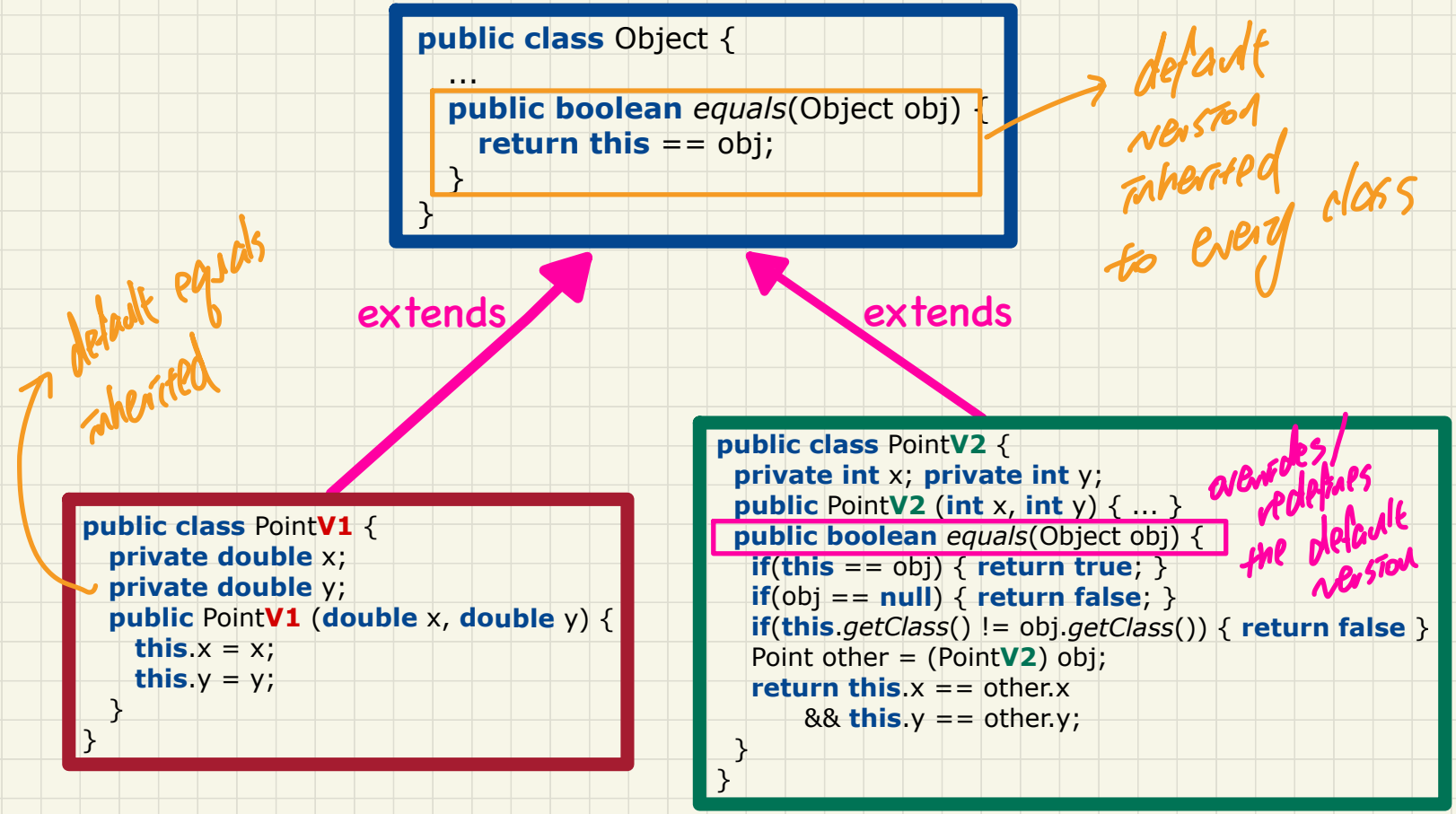
② x, y are ref vars.

c.o.

x.equals(y)

① x, y are ref var

The equals Method: To Override or Not?



The equals Method: Default Version

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

```
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 */
```

extends

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

p1 → PointV1 { x: 2, y: 3 }

p2 → PointV1 { x: 2, y: 3 }

p3 → PointV1 { x: 4, y: 6 }

① looks down: p1 == S (F)

② p1 == S

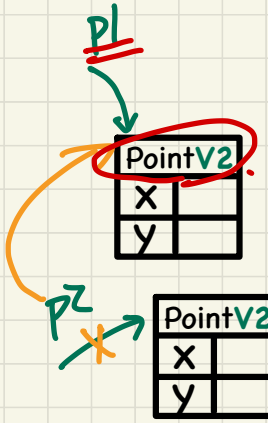
The equals Method: Overridden Version

Phase 1

```
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;
  }
}
```



PointV2 p1 = new PointV2(..);

p1.equals(..);

p1 is of type PointV2

↳ overridden version is called

p2 = p1;

p1.equals(p2);

same equals meth. is overridden; call that version.

The equals Method: Overridden Version

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

Have we missed:

p1 ~> null

p2 ~> null

if (this == null && obj == null) { return true }

no need to consider this == null :(a NPE would've occurred

reaching this line means this != obj

p1.equals(p2)

Scenario 1: move to phase 2

p1 != p2 and p2 is not null

p1 → PointV2 (x, y)

p2 → PointV2 (x, y)

Scenario 2

p1 → PointV2 (x, y)

(a non-null object is not equal to a null obj)

p2 → null

PointV2 (x, y)

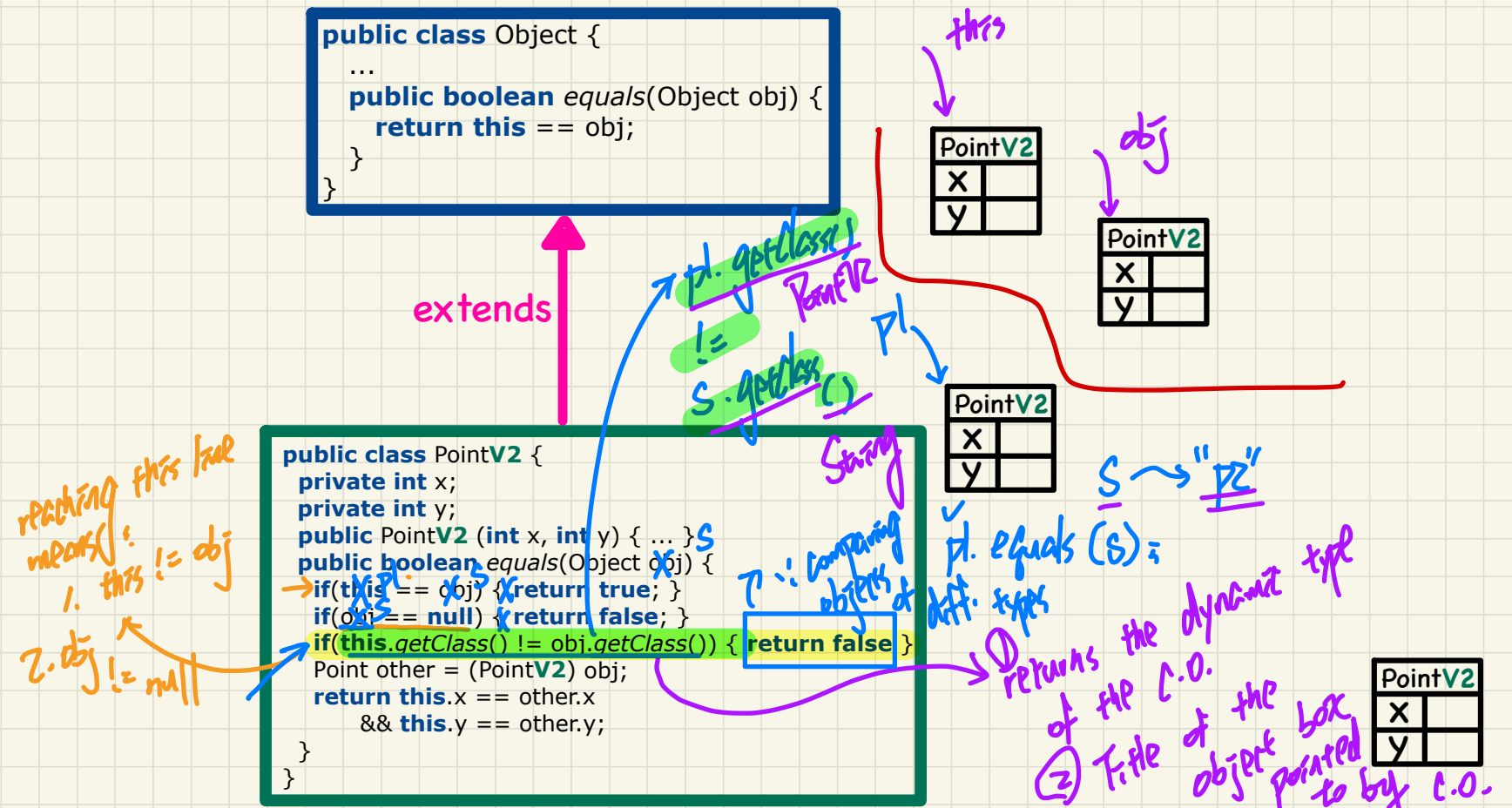
The equals Method: Overridden Version

Phase 3

```
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;
    }
}
```



The equals Method: Overridden Version

Phase 4

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

extends

means:

```
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;
  }
}
```

this

obj

static type
↳ restricts the range of methods that can be called on obj

this.x == obj.x

&& this.y == obj.y

① we can only invoke methods declared in the ST of obj.

② x, y only declared in PointV2

reaching this line means:

① this != obj

② obj != null

③ comparing objects of the same type

review up to this line by Monday