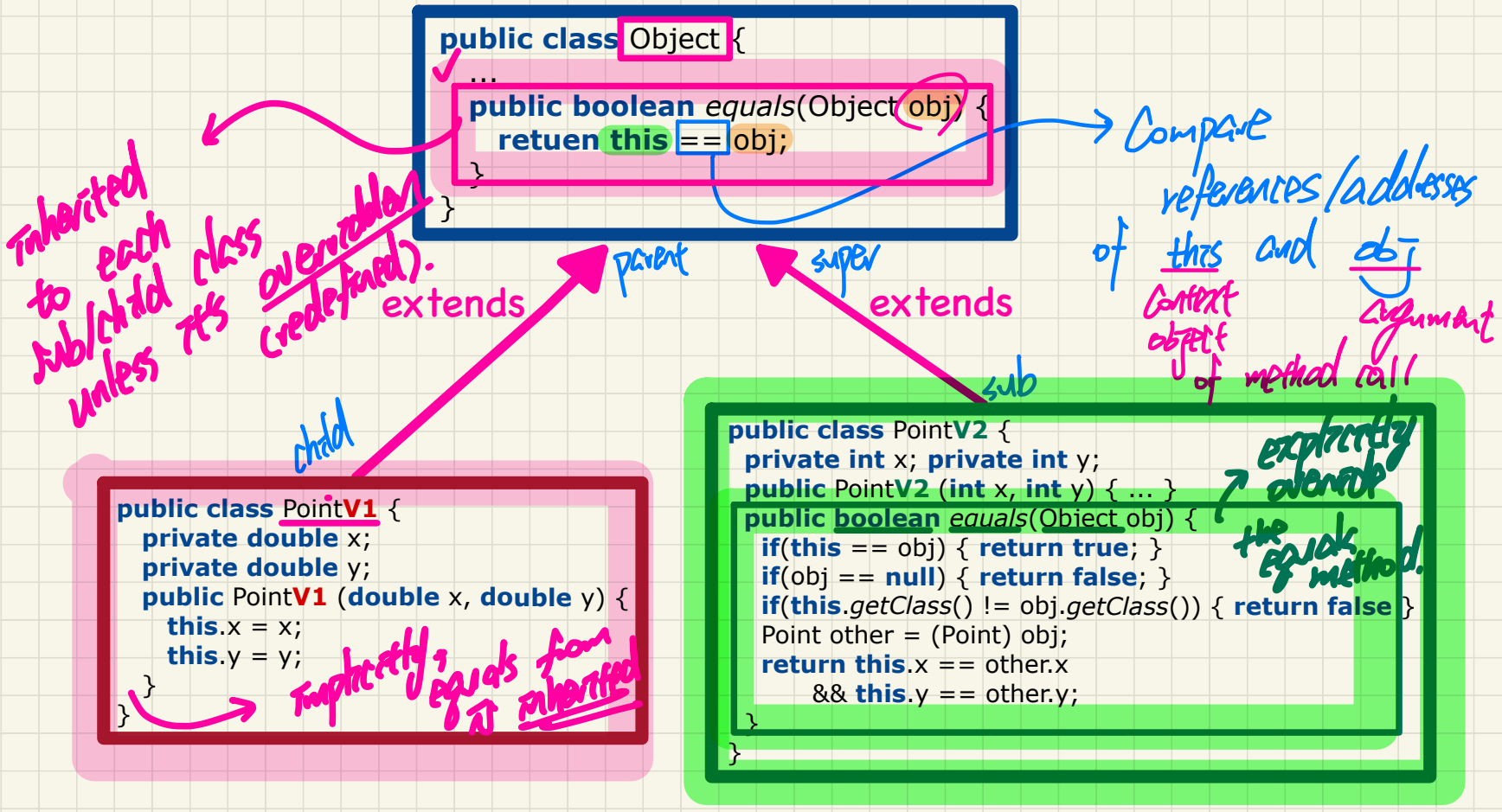
Lecture 3

Part A

Object Equality –
To Override or Not to Override

The equals Method: To Override or Not?



Lecture 3

Part B

Object Equality – Version 1: Default equals method

The equals Method: Default Version

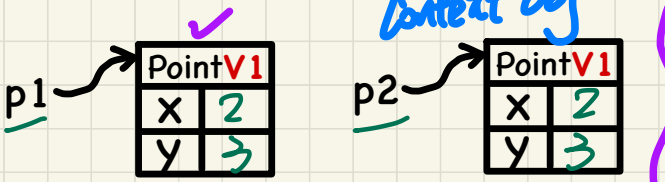
```
public class Object {
  ...
  public boolean equals(Object obj) {
    retuen this == obj;
  }
}
```

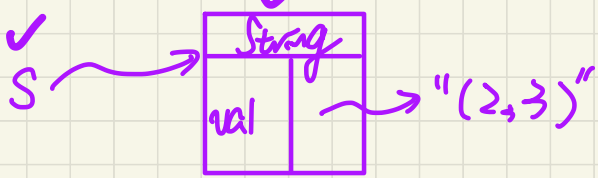
```
public class PointV1 {
  private int x;
  private int y;
  public PointV1 (int x, int y) {
    this.x = x;
    this.y = y;
  }
}
```

inherited

extends

```
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 */
```





context obj

argument

boils down:

p1 == s

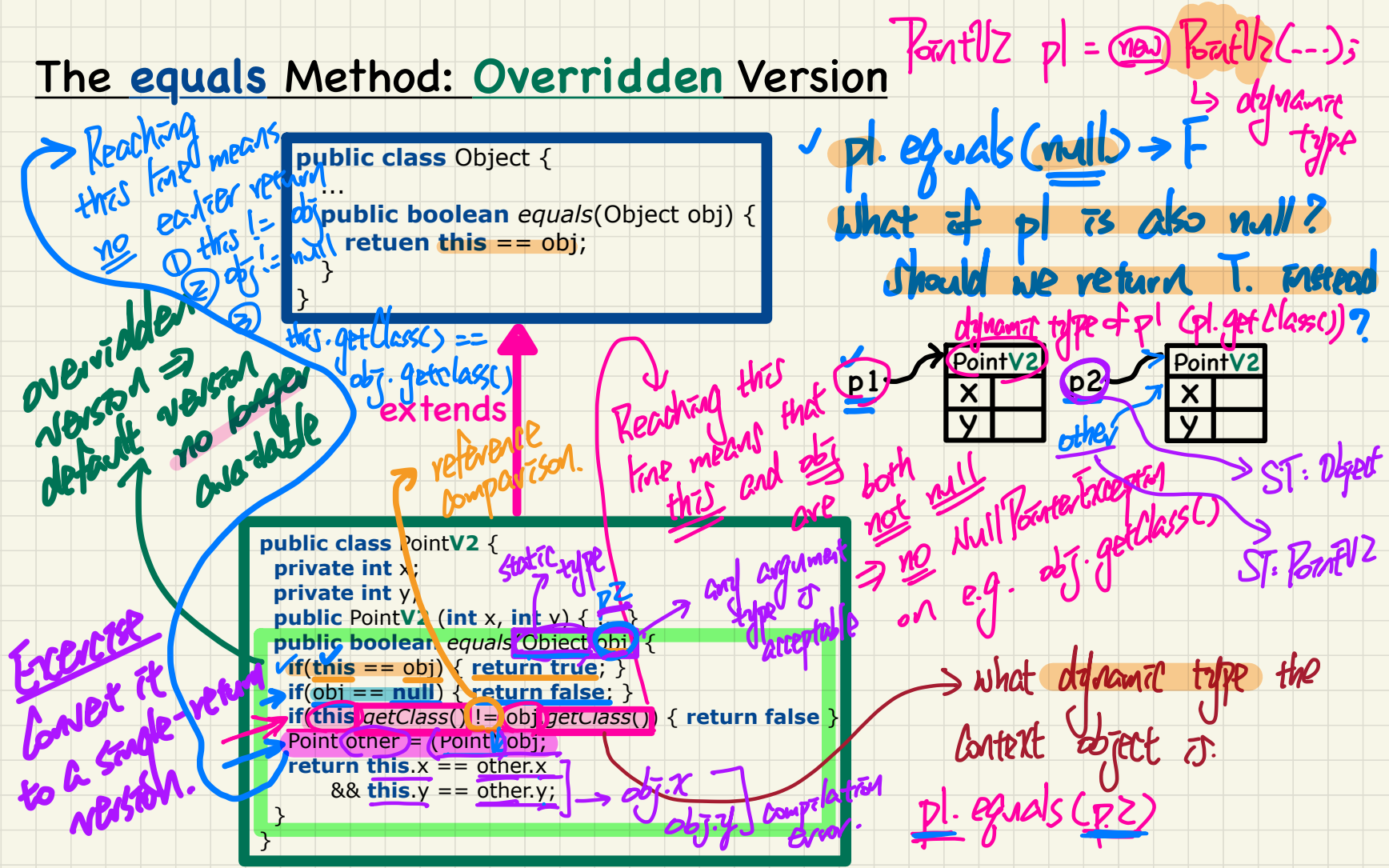
↳ writing this directly ⇒ Compilation error!!

Lecture 3

Part C

Object Equality – Version 2: Overridden equals method

The equals Method: Overridden Version



The equals Method: Overridden Version

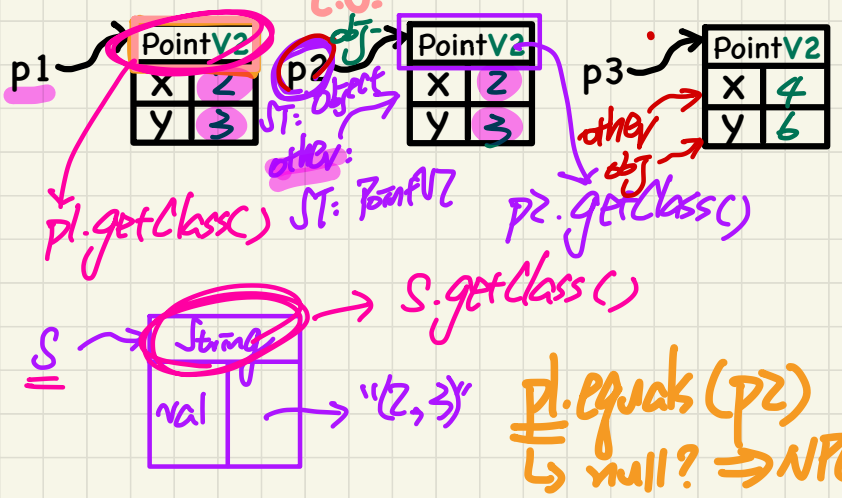
Example 1

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

```
1  String s = "(2, 3)";
2  PointV2 p1 = new PointV2(2, 3);
3  PointV2 p2 = new PointV2(2, 3);
4  PointV2 p3 = new PointV2(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));   /* true */
11 System.out.println(p2.equals(p3));   /* false */
```



The equals Method:

To Override or Not?

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

extends

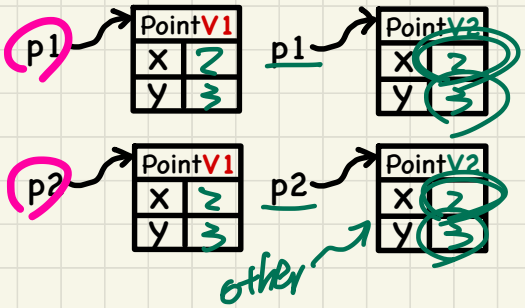
extends

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

```
public class PointV2 {
   private int x; double y;
   public 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;
   }
}
```

```
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 */
```

```
1  String s = "(2, 3)";
2  PointV2 p1 = new PointV2(2, 3);
3  PointV2 p2 = new PointV2(2, 3);
4  PointV2 p3 = new PointV2(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));   /* true */
11 System.out.println(p2.equals(p3));   /* false */
```

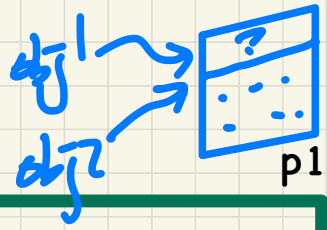


The equals Method: Overridden Version

Example 2

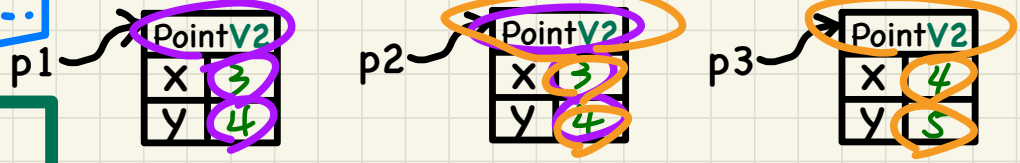
```
public class Object {
   ...
   public boolean equals(Object obj) {
      retuen this == obj;
   }
}
```

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 = (Point) 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 */
```



(A) Two objects are reference-equal.

(B) Two objects are contents-equal.

- If (A) is true, then (B) is true.
- X If (B) is true, then (A) is true.

not necessarily true ⇒ p1 vs. p2.

Lecture 3

Part D

Object Equality –
assertSame vs. assertEquals in JUnit

assertEquals: Reference Comparison or Not

assertEquals(exp1, exp2)

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

reference types

depending on the dynamic type of C.O. (exp1) overrides the equals method.

Case 1: If equals is ***not*** explicitly overridden in exp1's declared type

≈ ***assertSame***(exp1, exp2)

exp1 == exp2 (default dynamic in Object class)

```
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 ▭▭▭▭ objects */
```

p1.equals(p2) ⇒ p1==p2

p2.equals(p3) ⇒ p2==p3

Case 2: If equals is explicitly ***overridden*** in exp1's declared type

≈ exp1.**equals**(exp2)

customized version in dynamic exp1's dynamic type

```
PointV1 p1 = new PointV1(3, 4);
PointV1 p2 = new PointV1(3, 4);
PointV2 p3 = new PointV2(3, 4);
assertEquals(p1, p2); × /* ≈ p1.equals(p2) ≈ p1 == p2 */
assertEquals(p2, p3); × /* ≈ p2.equals(p3) ≈ p2 == p3 */
assertEquals(p3, p2); × /* ≈ p3.equals(p2) ≈ p3.getClass()==p2.getClass() */
```

p1.equals(p2) ⇒ p1 == p2 (default from object)

p2.equals(p3) ⇒ p2 == p3 (default from object)

p3.equals(p2);
↳ false. ↗ (!=) p2.getClass()
↳ p3.getClass() (default from object)

Lecture 3

Part E

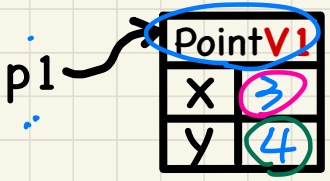
Object Equality – Asserting Reference vs. Object Equality

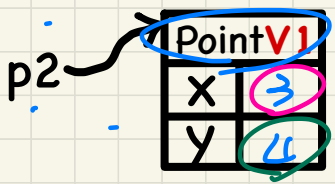
Testing Default Equality of Points in JUnit

```
@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.getX() == p2.getX() && p1.getY() == p2.getY());
}
```

p1 == p2 ⇒ False (default from object())

p2 == p1 ⇒ False






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

extends

```
public class PointV1 {
  private int x;
  private int y;
  public PointV1 (int x, int 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, p3); */ /* both fail */
  assertTrue(p3.equals(p4)); assertTrue(p4.equals(p3));
  assertEquals(p3, p4); assertEquals(p4, p3);
}
```

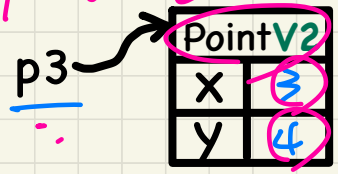
d.t.

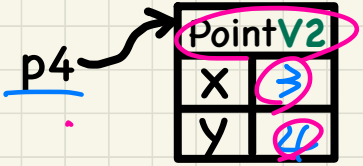
d.t.

p3.equals(p4)

overridden version from PointV2

p4.equals(p3)






```
public class Object {
  ...
  public boolean equals(Object obj) {
    retuen 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 = (Point) obj;
    return this.x == other.x
        && this.y == other.y;
  }
}
```

Testing Equality of Points in JUnit: Default vs. Overridden

```
@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 PointV2 is called */
  assertFalse(p2.equals(p1));
}
```

p1==p2

p1==p2 False. (default from Object).

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

extends

extends

p1

PointV1	
x	3
y	4

p2

PointV2	
x	3
y	4

p2.getClass() != p1.getClass() ⇒ False

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

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

Lecture 3

Part F

Object Equality –
Short-Circuit Effect of && and ||

Short-Circuit Evaluation: &&

logical conjunction

Left Operand op1	Right Operand op2	op1 && op2
true	true	true
true	false	false
false	true	false
false	false	false

Test Inputs:

x = 0, y = 10

x = 5, y = 10

```
System.out.println("Enter x:");
int x = input.nextInt();
System.out.println("Enter y:");
int y = input.nextInt();
if(x != 0 && y / x > 2) {
    System.out.println("y / x is greater than 2");
}
else { /* !(x != 0 && y / x > 2) == (x == 0 || y / x <= 2) */
    if(x == 0) {
        System.out.println("Error: Division by Zero");
    }
    else {
        System.out.println("y / x is not greater than 2");
    }
}
```

guarding constraint. should not be zero to avoid division-by-zero error.

x = 0, y = 10: `x != 0` is F, so `y / x > 2` is bypassed.

x = 5, y = 10: `x != 0` is T, `10 / 5 > 2` is 2 > 2 is F.

op1 && op2

if op1 is known to be false, it does not matter what op2 evaluates to ∴ evaluation of op2 can be bypassed.

Short-Circuit Evaluation: ||

logical disjunction

Left Operand op1	Right Operand op2	op1 || op2
false	false	false
true	false	true
false	true	true
true	true	true

Test Inputs:

x = 0, y = 10

x = 5, y = 10

```
System.out.println("Enter x:");
int x = input.nextInt();
System.out.println("Enter y:");
int y = input.nextInt();
if (x == 0 || y / x > 2) {
    if(x == 0) {
        System.out.println("Error: Division by Zero");
    }
    else {
        System.out.println("y / x is greater than 2");
    }
}
else { /* !(x == 0 || y / x > 2) == (x != 0 && y / x <= 2) */
    System.out.println("y / x is not greater than 2");
}
```

x: 0, 5 — y: 10

x == 0 || y / x > 2 → 0==0 || 10/0 > 2 → T (evaluation bypassed)

x == 0 → 5==0 → F

y / x > 2 → 10/5 > 2 → T

→ F.

guarding statement

op1 || op2

if op1 is known to be true, it does not matter what op2 is ∴ op2's evaluation bypassed

Short-Circuit Evaluation: Common Errors

order of G.C. is crucial

Test Inputs:
x = 0, y = 10

op1 && op2
op1 || op2
] evaluation at runtime is from left to right

Short-Circuit Evaluation is not exploited: crash when x == 0

```
if (y / x > 2 && x != 0) {
  /* do something */
}
else {
  /* print error */ }
```

→ 10/0 > 2 && 0 != 0
↓ division by zero error
G.C. useless

Short-Circuit Evaluation is not exploited: crash when x == 0

```
if (y / x <= 2 || x == 0) {
  /* print error */
}
else {
  /* do something */ }
```

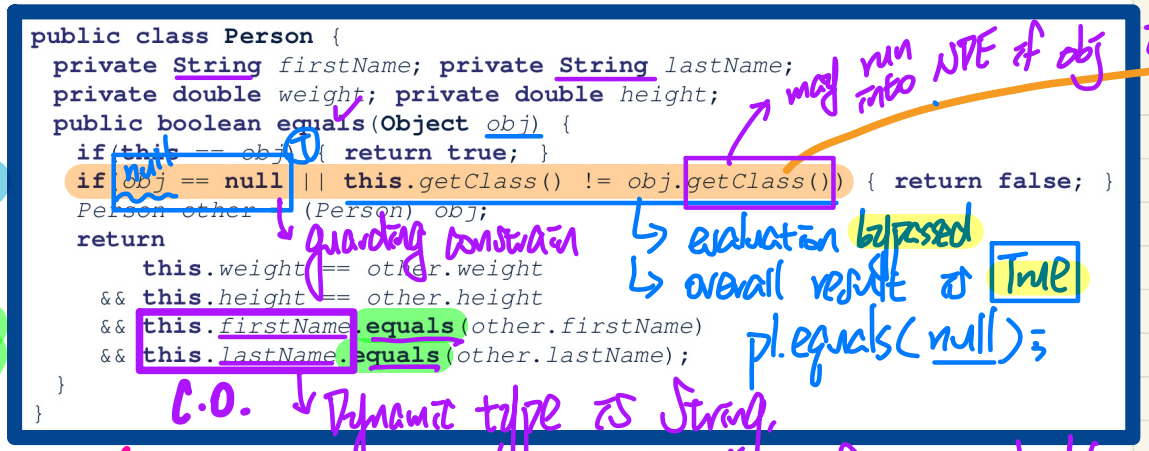
→ 10/0 <= 2 || 0 == 0
↓ division by zero error
G.C. useless

Lecture 3

Part G

Object Equality –
Exercises on the equals method

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



```
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 }
```


Exercise: Why is this equivalent the earlier version with two if-statements?

Q1: At Line 6, will there be a **NullPointerException** if obj == null?

Q2: At Line 6, what if we change it to:

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

short-circuit effect

evaluating this first, if obj is null, will result in NPE.

Evaluation at runtime is from L to R.

↳ g.c. not useful

Q3: At Lines 11 & 12 which version of the equals method is called?

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() { ... }
}
```

v3

```
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 }
```

D.T. Person.

Context object: dynamic type? Person

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

v2

v1: equals from Object class

```
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 }
```

v2

Lecture 3

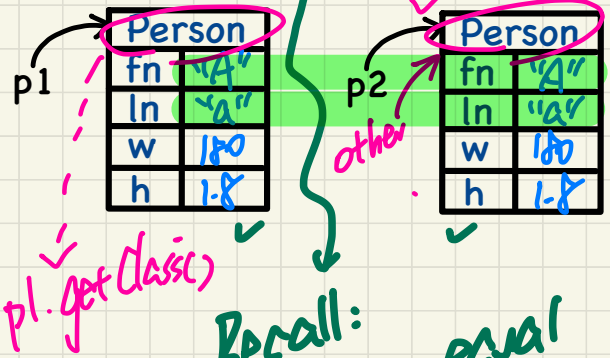
Part H

Object Equality -
Equality of Array-Typed Attribute

Testing Equality of Person/PersonCollector in JUnit (1)

```
@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));
```

Exercise.
How are these two assertions passed differently?



p1.getClass()

Recall:
Being Reference equal implies Contents equal.

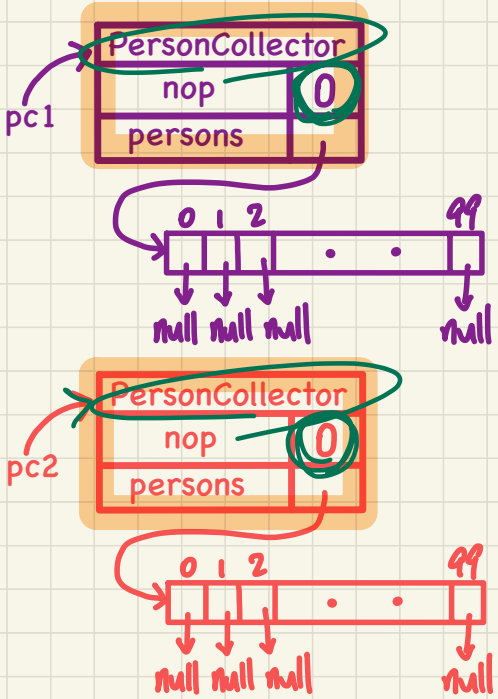
```
public class Person {
  private String firstName; private String lastName;
  private double weight; private double height;
  public boolean equals(Object obj) {
    if(this == obj) return true;
    if(obj == null || this.getClass() != obj.getClass()) return false;
    Person other = (Person) obj;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

p1 == p2 ✗ p3 == p4 ✓ p1 != p2 ✗

String

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;
}
```

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));
```

false

F

Person.equals

PC

Person

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;
}
```

pc1, pc2

0

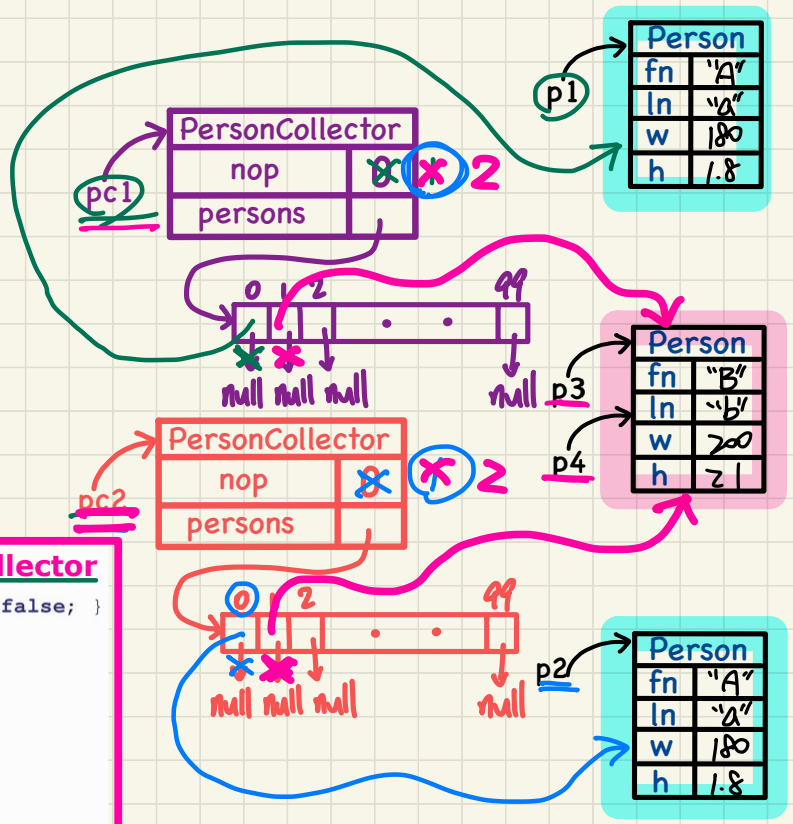
0

pc1

pc2

Person version

True.



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));
```

PersonColle. Person.

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	equal
0	T
1	T
2	F

p1

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

p3 / p4

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

p2

Person	
fn	"B"
ln	"b"
w	200
h	2

pc1

PersonCollector	
nop	2
persons	

pc2

PersonCollector	
nop	2
persons	

0 1 2 ... 99

null null

Person	
fn	"A"
ln	"a"
w	165
h	1.55