

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 }
```

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)`

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() { ... }  
}
```

```
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 }
```

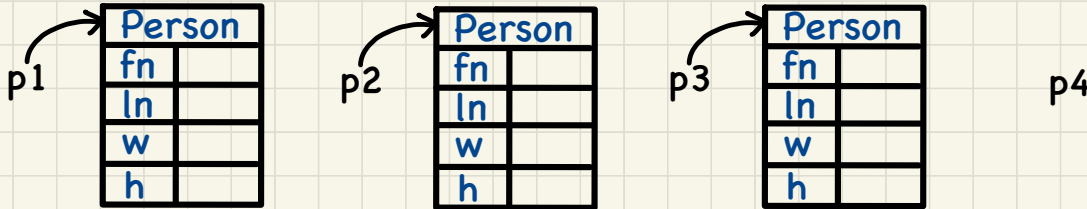
Q: At Line 9 of **PersonCollector**'s **equals** method which version of the **equals** method is called?

```
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)

@Test

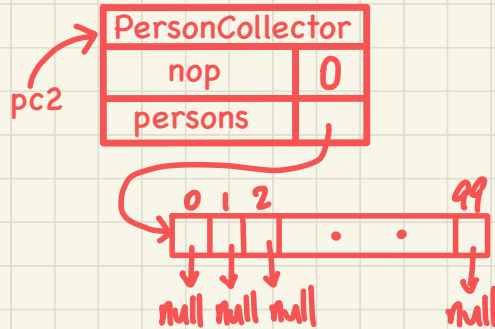
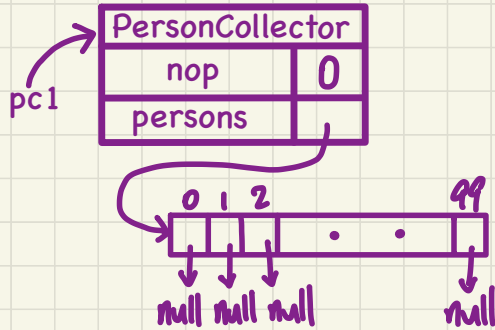
```
public void testPersonCollector() {  
    Person p1 = new Person("A", "a", 180, 1.8);  
    Person p2 = new Person("A", "a", 180, 1.8);  
    Person p3 = new Person("B", "b", 200, 2.1);  
    Person p4 = p3;  
    assertFalse(p1 == p2); assertTrue(p1.equals(p2));  
    assertTrue(p3 == p4); assertTrue(p3.equals(p4));  
}
```



```
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;  
}
```

Testing Equality of Person/PersonCollector in JUnit (3)

(continued from [testPersonCollector](#))

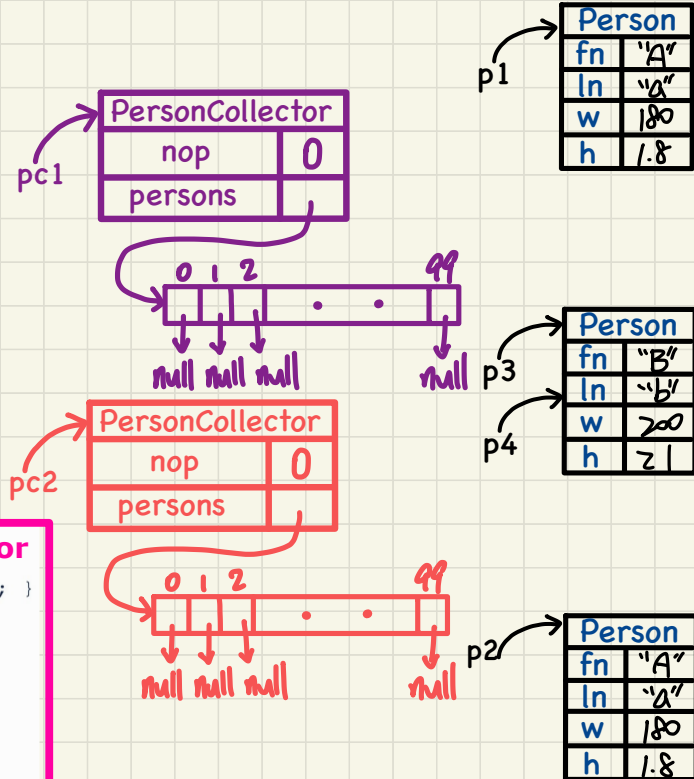
```
pc1.addPerson(p1);
assertFalse(pc1.equals(pc2));
pc2.addPerson(p2);
assertFalse(pc1.getPersons()[0] == pc2.getPersons()[0]);
assertTrue(pc1.getPersons()[0].equals(pc2.getPersons()[0]));
assertTrue(pc1.equals(pc2));
pc1.addPerson(p3);
pc2.addPerson(p4);
assertTrue(pc1.getPersons()[1] == pc2.getPersons()[1]);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

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



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));
```

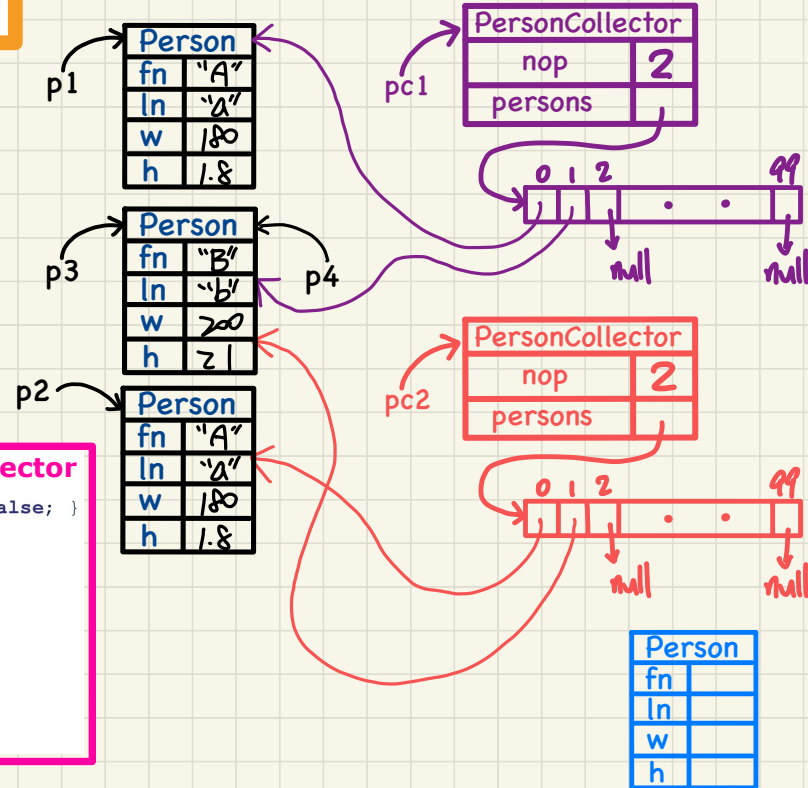
Person	
fn	
ln	
w	
h	

```
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



Inheritance: Motivating Problem

Nouns -> classes, attributes, accessors

Verbs -> mutators

Problem: A student management system stores data about students. There are two kinds of university students: resident students and non-resident students. Both kinds of students have a name and a list of registered courses. Both kinds of students are restricted to register for no more than 10 courses. When calculating the tuition for a student, a base amount is first determined from the list of courses they are currently registered (each course has an associated fee). For a non-resident student, there is a discount rate applied to the base amount to waive the fee for on-campus accommodation. For a resident student, there is a premium rate applied to the base amount to account for the fee for on-campus accommodation and meals.

First Design Attempt

```
public class Student {  
    private Course[] courses;  
    private int noc;  
  
    private int kind;  
    private double premiumRate;  
    private double discountRate;  
  
    public Student (int kind){  
        this.kind = kind;  
    }  
    ...  
}
```

```
public double getTuition(){  
    double tuition = 0;  
    for(int i = 0; i < this.noc; i++){  
        tuition += this.courses[i].fee;  
    }  
    if (this.kind == 1) {  
        return tuition * this.premiumRate;  
    }  
    else if (this.kind == 2) {  
        return tuition * this.discountRate;  
    }  
}
```

```
public void register(Course c){  
    int MAX = -1;  
    if (this.kind == 1) { MAX = 6; }  
    else if (this.kind == 2) { MAX = 4; }  
    if (this.noc == MAX) { /* Error */ }  
    else {  
        this.courses[this.noc] = c;  
        this.noc ++;  
    }  
}
```

First Design Attempt

```
public class Student {  
    private Course[] courses;  
    private int noc;  
  
    private int kind;  
    private double premiumRate;  
    private double discountRate;  
  
    public Student (int kind){  
        this.kind = kind;  
    }  
    ...  
}
```

```
public double getTuition(){  
    double tuition = 0;  
    for(int i = 0; i < this.noc; i++){  
        tuition += this.courses[i].fee;  
    }  
    if (this.kind == 1) {  
        return tuition * this.premiumRate;  
    }  
    else if (this.kind == 2) {  
        return tuition * this.discountRate;  
    }  
}
```

```
public void register(Course c){  
    int MAX = -1;  
    if (this.kind == 1) { MAX = 6; }  
    else if (this.kind == 2) { MAX = 4; }  
    if (this.noc == MAX) { /* Error */ }  
    else {  
        this.courses[this.noc] = c;  
        this.noc ++;  
    }  
}
```

Good design?

Judge by Cohesion

First Design Attempt

```
public class Student {  
    private Course[] courses;  
    private int noc;  
  
    private int kind;  
    private double premiumRate;  
    private double discountRate;  
  
    public Student (int kind){  
        this.kind = kind;  
    }  
    ...  
}
```

```
public double getTuition(){  
    double tuition = 0;  
    for(int i = 0; i < this.noc; i++){  
        tuition += this.courses[i].fee;  
    }  
    if (this.kind == 1) {  
        return tuition * this.premiumRate;  
    }  
    else if (this.kind == 2) {  
        return tuition * this.discountRate;  
    }  
}
```

```
public void register(Course c){  
    int MAX = -1;  
    if (this.kind == 1) { MAX = 6; }  
    else if (this.kind == 2) { MAX = 4; }  
    if (this.noc == MAX) { /* Error */ }  
    else {  
        this.courses[this.noc] = c;  
        this.noc ++;  
    }  
}
```

Good design?

Judge by Single Choice Principle

- **Repeated** if-conditions
- A new kind is **introduced**?
- An existing kind is **obsolete**?