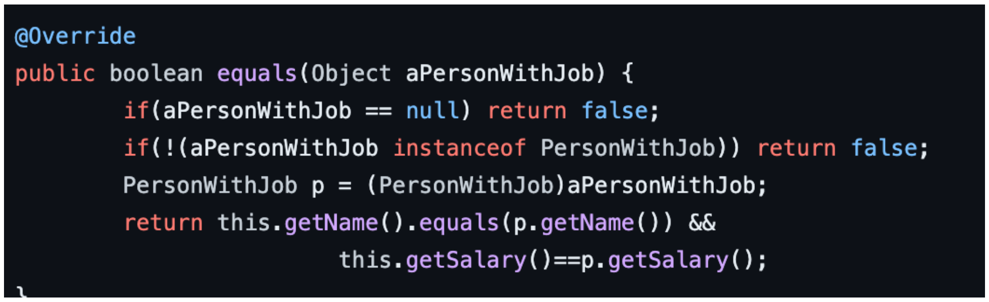
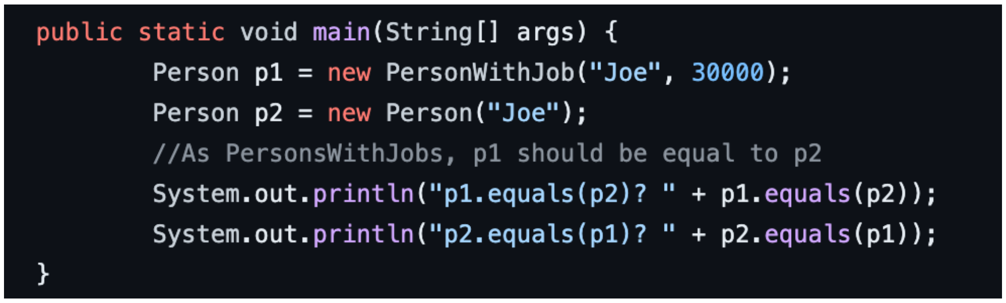
Explanation to why p1.equals(p2) is false and p2.equals(p1) is true in the initial solution provided.
Answer: To explain, lets analyze the initial overridden equals method in both Person and PersonWithJob class.



```java
@Override
public boolean equals(Object aPersonWithJob) {
        if(aPersonWithJob == null) return false;
        if(!(aPersonWithJob instanceof PersonWithJob)) return false;
        PersonWithJob p = (PersonWithJob)aPersonWithJob;
        return this.getName().equals(p.getName()) &&
                        this.getSalary()==p.getSalary();
```

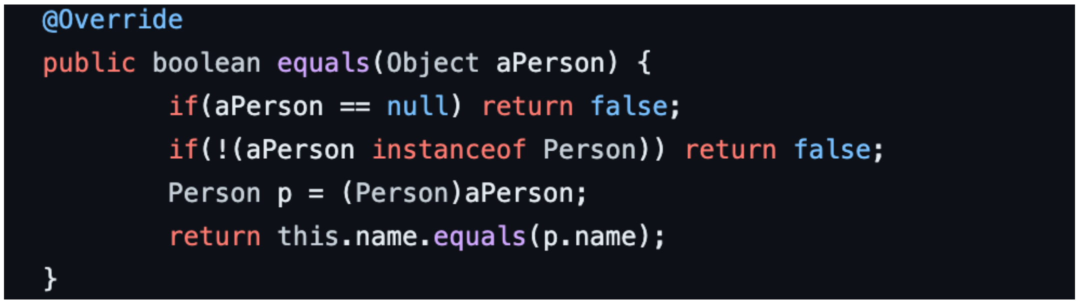
: Person With Job equals method



```java
 public static void main(String[] args) {
         Person p1 = new PersonWithJob("Joe", 30000);
         Person p2 = new Person("Joe");
         //As PersonsWithJobs, p1 should be equal to p2
         System.out.println("p1.equals(p2)? " + p1.equals(p2));
         System.out.println("p2.equals(p1)? " + p2.equals(p1));
 }
```

: Main method

In the **main** method, it's evident that **p1** holds an object of type **PersonWithJob**, which has an overridden **equals** method. However, **p2** is an instance of type **Person**. When we execute **p1.equals(p2)**, the **equals** method of **PersonWithJob** is invoked. Within the method, we check if the provided object is an instance of **PersonWithJob**. Since we are passing **p2** as an object of type **Person**, this condition fails, and **false** is returned.



```java
 @Override
 public boolean equals(Object aPerson) {
         if(aPerson == null) return false;
         if(!(aPerson instanceof Person)) return false;
         Person p = (Person)aPerson;
         return this.name.equals(p.name);
 }
```

: Person equals method

Now, in **p2.equals(p1)**, the **equals** method of the **Person** class is executed. Since **p1** belongs to **PersonWithJob**, which is a subclass of **Person**, the instance check is valid, and control moves to the next line. Since both objects have the same person name, **true** is returned."