

Q1

In the given code, the **PersonWithJob** class extends the **Person** class, and both classes override the **equals** method.

The **PersonWithJob** class's **equals** method checks if the provided object is an instance of **PersonWithJob**. In contrast, the **Person** class's **equals** method checks if the provided object is an instance of **Person**.

The image shows a Java code snippet and two class diagrams. The code defines a `main` method that creates two objects: `p1` of type `PersonWithJob` and `p2` of type `Person`. It then prints the results of `p1.equals(p2)` and `p2.equals(p1)`. The class diagram shows `Person` as a base class with methods `Person(String)`, `main(String[])`, `equals(Object)`, and a field `name`. `PersonWithJob` is a subclass of `Person` with methods `PersonWithJob(String, double)`, `main(String[])`, `equals(Object)`, and a field `salary`. Handwritten annotations include "calls" pointing from the `p1.equals(p2)` call to the `Person` class's `equals` method, and "Calls" pointing from the `p2.equals(p1)` call to the `PersonWithJob` class's `equals` method. A handwritten note at the bottom explains the results: `p1.equals(p2)` compares if `p2` is an object of `PersonWithJob` (which is not), but `p2.equals(p1)` compares if `p1` is an instance of `Person` (which is yes).

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

they give different results, $P_1.equals(P_2)$: compares if P_2 is an object of `PersonWithJob` which is not, but $P_2.equals(P_1)$: compares if P_1 is instance of `Person` which is yes.

The composition solution in the zip folder

Q2