perplexity

You're correctly overriding the sound() method in the Dog class using @Override. But in your main() method, you're not using the overridden method — you're calling the method on an object of the base class (Animal), not the subclass (Dog).

Here's your code:
java
CopyEdit

```
Animal animal = new Animal(); // Base class object
Dog dog = new Dog(); // Subclass object
animal.sound(); // Calls Animal's sound()
```

This line:
java
CopyEdit

```
animal.sound(); // Prints: Animal makes a sound
```

does not call the Dog class method because animal is an instance of Animal, not Dog.

If you want to see the overridden method in action:
✅ Option 1: Call sound() on the dog object
java
CopyEdit

```
dog.sound(); // This will print: Dog barks
```

✅ Option 2: Use upcasting with polymorphism
java
CopyEdit

```
Animal animal = new Dog(); // Animal reference, Dog object
animal.sound(); // This will now call Dog's overridden method: Dog barks
```

Final Correct Example:
java
CopyEdit

```
class Animal {
void sound() {
System.out.println("Animal makes a sound");
}
}
```

```
class Dog extends Animal {
@Override
void sound() {
System.out.println("Dog barks");
}
}

class Main {
public static void main(String[] args) {
Animal animal = new Dog(); // Upcasting
animal.sound(); // Output: Dog barks
}
} give me same
```

Certainly! Here's the same explanation and example code you requested:

You're correctly overriding the `sound()` method in the `Dog` class using `@Override`. But in your `main()` method, you're not using the overridden method — you're calling the method on an object of the base class (`Animal`), not the subclass (`Dog`).

Here's your code:

```
Animal animal = new Animal(); // Base class object
Dog dog = new Dog();          // Subclass object
animal.sound();               // Calls Animal's sound()
```

This line:

```
animal.sound(); // Prints: Animal makes a sound
```

does not call the `Dog` class method because `animal` is an instance of `Animal`, not `Dog`.

If you want to see the overridden method in action:

✅ Option 1: Call `sound()` on the `dog` object

```
dog.sound(); // This will print: Dog barks
```

✅ Option 2: Use upcasting with polymorphism

```
Animal animal = new Dog(); // Animal reference, Dog object
animal.sound(); // This will now call Dog's overridden method: Dog barks
```

Final Correct Example:

```
class Animal {
    void sound() {
        System.out.println("Animal makes a sound");
    }
}

class Dog extends Animal {
    @Override
    void sound() {
        System.out.println("Dog barks");
    }
}

class Main {
    public static void main(String[] args) {
        Animal animal = new Dog(); // Upcasting
        animal.sound(); // Output: Dog barks
    }
}
```

If you run this, the output will be:

```
Dog barks
```

This demonstrates runtime polymorphism in Java, where the overridden method in the subclass is called even when using a superclass reference pointing to a subclass object.