Overriding Method Access Modifiers

Yes, you are **absolutely right**, and this is an **important rule in Java method overriding**.

✅ Rule:

> The **access modifier** of an overriding method **must be the same or more accessible (wider)** than the method being overridden.

🔓 Access Modifier Levels (from most restrictive to most accessible):

Modifier	Accessible To
`private`	Only within the same class
(default)	Only within the same package
`protected`	Same package + subclasses (even in other packages)
`public`	Accessible from anywhere

📘 Explanation:

When you override a method, the subclass version **should not restrict** access more than the superclass method. Otherwise, it would **break the principle of polymorphism**, where a subclass should be usable wherever its superclass is.

❌ Invalid Example (Compiler Error):

```java

class Parent {
    protected void display() {
        System.out.println("Parent");
```

```
    }
}

class Child extends Parent {
    private void display() { // ❌ Error: can't reduce visibility
        System.out.println("Child");
    }
}
```

✅ Valid Examples:

1. Same access modifier:

```
java

class Parent {
    protected void display() { }
}

class Child extends Parent {
    protected void display() { } // ✅ OK
}
```

2. Wider access modifier:

```
java

class Parent {
    protected void display() { }
}

class Child extends Parent {
    public void display() { } // ✅ OK: public is more accessible than protected
}
```

❗ Note:

- If the parent method is `private`, it is **not visible** to the child, so **no overriding** happens — it's a **new method** in the child.
- The compiler will catch access modifier violations during override.

🎯 Conclusion:

> **"Access modifier of the overriding method must be same or more accessible (wider) than that of the overridden method."**

Let me know if you want a memory trick, diagram, or interview question on this!