

In Java, visibility control is managed through access modifiers, which determine the accessibility of classes, methods, and variables. When it comes to overriding methods in derived (subclass) classes, there are specific rules about visibility that you need to follow.

#### Access Modifiers in Java

**public:** The member is accessible from any other class.

**protected:** The member is accessible within its own package and by subclasses.

**default (no modifier):** The member is accessible only within its own package.

**private:** The member is accessible only within its own class.

#### Visibility Control in Derived Methods

When you override a method in a subclass, you cannot reduce the visibility of the inherited method. This means:

You can maintain the same visibility or increase it, but not decrease it.

This rule ensures that the subclass can be used in place of the superclass without violating the access expectations of the superclass's clients.

#### Examples

##### Valid Override

Java

```
1class SuperClass {
2    protected void display() {
3        System.out.println("SuperClass display");
4    }
5}
6
7class SubClass extends SuperClass {
8    @Override
9    public void display() { // Increased visibility
10        System.out.println("SubClass display");
11    }
12}
```

In this example, the display method in SubClass is overridden with increased visibility from protected to public, which is allowed.

##### Invalid Override

Java

```
1class SuperClass {
2    public void display() {
3        System.out.println("SuperClass display");
4    }
5}
6
7class SubClass extends SuperClass {
8    // This will cause a compile-time error
9    @Override
10    protected void display() { // Attempting to reduce visibility
11        System.out.println("SubClass display");
12    }
13}
```

In this example, trying to override the display method with reduced visibility from public to protected will result in a compile-time error.

#### Visibility Control in Java

**Classes:** Top-level classes can only be public or package-private (default). Inner classes can have all four access levels.

**Methods and Variables:** Can have any of the four access levels. The choice of access level affects encapsulation and the ability to change implementations without affecting other parts of the program.

#### Best Practices

Use the most restrictive access level that makes sense for a particular member. This helps in encapsulating the details and reducing the impact of changes.

**Public API:** Methods that are part of the public API should be public and well-documented.

**Internal Implementation:** Methods and variables that are part of the internal implementation should be private or protected to prevent unintended use.

By understanding and applying these rules, you can effectively control the visibility of methods and variables in Java, ensuring proper encapsulation and adherence to object-oriented

principles.