ACCESS MODIFIERS

✅ **POLISHED VERSION:**

> In Java, **access modifiers** define the **visibility** of classes, methods, and variables.

OR,
Access modifiers are the rules that control who can see or use parts of your code - like variables, methods, or classes.
They allow us to **control whether a method or variable can be accessed from other classes, packages, or only within the same class**.

✅ **REFERENCE: `PUBLIC` ACCESS MODIFIER**

> `public` means the **variable, method, or class is accessible from anywhere** — across **classes, packages, and even subclasses**.

✅ **1. Same Class — ✔️ Allowed**

```
public class Demo {
    public int number = 10;

    public void show() {
        System.out.println(number); // ✅ Accessing directly inside the same class
    }
}
```

✅ **2. Same Package — Subclass — ✔️ Allowed**

📁 **`packageOne/Demo.java`**

```
package packageOne;

public class Demo {
    public int number = 100;
}
```

📁 `packageOne/Child.java`

```
package packageOne;

public class Child extends Demo {
    public void printNumber() {
        System.out.println(number); // ✅ Public member is accessible
    }
}
```

✅ **3. SAME PACKAGE — NON-SUBCLASS — ✔️ ALLOWED : TWO CLASSES, BUT THEY ARE NOT CONNECTED THROUGH INHERITANCE. THAT'S WHAT WE MEAN BY "NON-SUBCLASS."**

💡 **KEY CONCEPT:**

> A **subclass** is a class that uses `extends` to inherit from another class.
> A **non-subclass** is just a **completely separate class**, not related through inheritance.

📁 **packageOne/Demo.java**

```
package packageOne;

public class Demo {
    public int number = 200;
}
```

📁 **packageOne/Test.java**

```
package packageOne;

public class Test {
    public static void main(String[] args) {
        Demo d = new Demo();          // ✅ Create object of Demo
        System.out.println(d.number); // ✅ Access public variable
    }
}
```

🔍 **WHAT'S HAPPENING?**

- ✅ **Same package** → `packageOne`
- ✅ Two separate classes → `Demo` and `Test`
- ❌ `Test` **is not a subclass** of `Demo` (no `extends`)
- ✅ Still works because the variable is `public`

✅ **4. DIFFERENT PACKAGE — SUBCLASS — ✔️ ALLOWED**

📁 **`packageOne/Demo.java`**

```
package packageOne;

public class Demo {
    public int number = 50;
}
```

📁 `packageTwo/Child.java`

```
package packageTwo;

import packageOne.Demo;

public class Child extends Demo {
    public void display() {
        System.out.println(number); // ✅ Accessible even across packages
```

```
    }
}
```

📌 WHEN DO WE USE `IMPORT` IN JAVA?

	Scenario	Do You Need import?
1	Classes are in the **same package**	❌ No `import` needed
2	One class is in a **different package**	✅ Yes, you need to `import`

✅ 5. DIFFERENT PACKAGE — NON-SUBCLASS — ✔️ ALLOWED (ONLY FOR `PUBLIC`)

📁 Example Code — Different Packages, No Inheritance

📂 **`packageOne/Demo.java`**

```
package packageOne;

public class Demo {
    public int number = 300;
}
```

📂 **`packageTwo/Test.java`**

```
package packageTwo;

// ✅ We must import Demo class from another package
import packageOne.Demo;

public class Test {
    public static void main(String[] args) {
        Demo d = new Demo();               // ✅ Creating object
        System.out.println(d.number);      // ✅ Accessing public variable
    }
}
```

✅ WHY THIS WORKS:

- `Demo` is in **packageOne**
- `Test` is in **packageTwo**
- `Test` is **not a subclass** of `Demo` — it's just using an object
- The variable `number` is marked `public`, so it can be accessed across packages

NOTE : start a mini-project using all modifiers!

🛡️ `PRIVATE` ACCESS MODIFIER IN JAVA

🔒 `PRIVATE` ACCESS MODIFIER IN JAVA

> `private` means: **Accessible only within the same class**.
> You **cannot access it from outside**, not even from subclasses — not even in the same package.

✅ **SCENARIO 1: SAME CLASS — ✔️ ALLOWED**

```
public class Demo {
    private int secretCode = 1234;

    public void displayCode() {
        System.out.println(secretCode); // ✅ Works fine here
    }
}
```

❌ **SCENARIO 2: SAME PACKAGE – SUBCLASS — ❌ NOT ALLOWED**

```
// File: packageOne/Demo.java
package packageOne;

public class Demo {
    private int secretCode = 1234;
}
```

```
// File: packageOne/Child.java
package packageOne;

public class Child extends Demo {
    public void show() {
        System.out.println(secretCode);    // ❌ Compilation Error: secretCode has
                                                private access//
    }
}
```

❌ **SCENARIO 3: SAME PACKAGE – NON-SUBCLASS — ❌ NOT ALLOWED**

```
// File: packageOne/Demo.java
package packageOne;

public class Demo {
    private int pin = 5555;
}
```

```
// File: packageOne/Test.java
package packageOne;

public class Test {
    public static void main(String[] args) {
        Demo d = new Demo();
        System.out.println(d.pin); // ❌ Error: pin has private access
    }
}
```

❌ **SCENARIO 4: DIFFERENT PACKAGE – SUBCLASS — ❌ NOT ALLOWED**

```
// File: packageOne/Demo.java
package packageOne;
```

```java
public class Demo {
    private int id = 999;
}
```

```java
// File: packageTwo/Child.java
package packageTwo;

import packageOne.Demo;

public class Child extends Demo {
    public void display() {
        System.out.println(id); // ❌ Error: id is private
    }
}
```

❌ SCENARIO 5: DIFFERENT PACKAGE – NON-SUBCLASS — ❌ NOT ALLOWED

```java
// File: packageOne/Demo.java
package packageOne;

public class Demo {
    private String status = "locked";
}
```

```java
// File: packageTwo/Test.java
package packageTwo;

import packageOne.Demo;

public class Test {
    public static void main(String[] args) {
        Demo d = new Demo();
        System.out.println(d.status); // ❌ Error: status is private
    }
}
```

🛡️ PROTECTED ACCESS MODIFIER IN JAVA

✅ Scenario 1: Same Class — ✔️ Allowed

```java
public class Demo {
    protected int code = 123;

    public void display() {
        System.out.println(code); // ✅ works
    }
}
```

✅ Scenario 2: Same Package – Subclass — ✔️ Allowed

```java
// File: packageOne/Demo.java
package packageOne;

public class Demo {
    protected int number = 777;
}
```

```java
// File: packageOne/Child.java
package packageOne;

public class Child extends Demo {
    public void show() {
        System.out.println(number); // ✅ allowed
    }
}
```

✅ Scenario 3: Same Package – Non-subclass — ✔️ Allowed

```java
// File: packageOne/Demo.java
package packageOne;

public class Demo {
    protected String status = "Active";
}
```

```java
// File: packageOne/Test.java
package packageOne;

public class Test {
    public static void main(String[] args) {
        Demo d = new Demo();
        System.out.println(d.status); // ✅ accessible within same package
    }
}
```

✅ Scenario 4: Different Package – Subclass — ✔️ Allowed

> Only if accessed **inside the subclass** using inheritance (not via object)

```java
// File: packageOne/Demo.java
package packageOne;

public class Demo {
    protected int value = 888;
}
```

```java
// File: packageTwo/Child.java
package packageTwo;

import packageOne.Demo;
```

```java
public class Child extends Demo {
    public void printValue() {
        System.out.println(value); // ✅ accessible because we're inside a subclass
    }
}
```

❌ Scenario 5: Different Package – Non-subclass — ❌ Not Allowed

```java
// File: packageOne/Demo.java
package packageOne;

public class Demo {
    protected int x = 99;
}
```

```java
// File: packageTwo/Test.java
package packageTwo;

import packageOne.Demo;

public class Test {
    public static void main(String[] args) {
        Demo d = new Demo();
        System.out.println(d.x); // ❌ Error: x has protected access in Demo
    }
}
```

🛡️ DEFAULT ACCESS MODIFIER IN JAVA

📘 WHAT IS `DEFAULT` ACCESS IN JAVA?

> If you **don't specify any access modifier**, Java automatically gives it **default/package-private access**.

🔑 VISIBILITY RULE:

```java
// default access = accessible only within the same package
```

✅ SCENARIO 1: SAME PACKAGE — ✔️ ALLOWED

```java
// File: packageOne/Test.java
package packageOne;

public class Test {
    public static void main(String[] args) {
        Demo d = new Demo();     // ✅ class is visible
        System.out.println(d.id); // ✅ variable is accessible
    }
}
```

❌ SCENARIO 2: DIFFERENT PACKAGE — ❌ NOT ALLOWED

```java
// File: packageTwo/Test.java
package packageTwo;

import packageOne.Demo;

public class Test {
    public static void main(String[] args) {
        Demo d = new Demo(); // ❌ Error: Demo is not public and not accessible here
    }
}
```

📊 SUMMARY — `DEFAULT` ACCESS

	Scenario	Access Allowed?	Reason
1	✅ Same class	✅ Yes	Always accessible
2	Same package – subclass	✅ Yes	Default allows full access in same package
3	Same package – non-subclass	✅ Yes	Still allowed — same package
4	Different package – subclass	❌ No	Default not visible outside package
5	Different package – non-subclass	❌ No	Not allowed

🎙️ Q1: EXPLAIN ALL FOUR ACCESS MODIFIERS IN JAVA WITH REAL EXAMPLES. HOW ARE THEY DIFFERENT FROM EACH OTHER IN TERMS OF PACKAGE AND INHERITANCE?

ANSWER:

Sure. In Java, we have four main access modifiers — `public`, `private`, `protected`, and `default` (which is also called *package-private*). They control how accessible a class, method, or variable is from other classes or packages.
Let me walk through each:

✅ `public`

- It's the most open access modifier.
- Anything marked `public` can be accessed from **anywhere** — same class, same package, different package, subclass, or non-subclass.
- **Example:** In my SSR automation project, I made common utility methods like `getDriver()` public, so they could be accessed across all classes and packages.

🔒 `private`

- This is the most restrictive. A `private` member can be accessed **only inside the same class**.
- Even subclasses in the same package **cannot access it**.
- **Example:** I used private variables like WebElement locators inside page classes to restrict direct access from outside and maintain encapsulation.

🛡️ `protected`

- This one's interesting — it's accessible:
  - Inside the same class ✅
  - Inside the same package ✅ (whether subclass or not)
  - In a **different package**, but **only if it's a subclass** and accessed via inheritance.
- If you try to access a `protected` member from a different package using an object, it won't work.
- **Example:** I used `protected` for reusable `waitUntilClickable()` method in a `BasePage`, which was inherited by multiple page classes across packages.

⚙️ `default` (no modifier)

- If we don't specify any modifier, Java treats it as **package-private**.
- It means the member is accessible to all classes **within the same package**, but **not visible outside**, even to subclasses.
- **Example:** I used default access for internal helper classes that weren't needed outside the package.

🎙️ Q2: WILL THIS CODE COMPILE AND RUN?

```
package packageOne;

public class Base {
    protected int count = 100;
}
```

```
package packageTwo;

import packageOne.Base;

public class Sub extends Base {
    public void display() {
        System.out.println(count);
    }
}
```

Answer:
Yes, this code will compile and run correctly.
That's because the variable `count` is marked as `protected`, and we're accessing it inside a **subclass (`Sub`)** that extends the parent class `Base`.
Even though the subclass is in a **different package**, `protected` allows access **through inheritance** — and that's exactly what's happening here.
So this scenario fits the rule:

> "In a different package, `protected` members are accessible only via inheritance, and only inside the subclass."

Since `count` is inherited and accessed directly within `Sub`, this works perfectly.

Q3. WHAT WILL HAPPEN IF YOU TRY TO ACCESS A `PRIVATE` METHOD FROM A SUBCLASS IN THE SAME PACKAGE?

Answer:
If we try to access a `private` method from a subclass, even within the same package, the code will **not compile**.
That's because `private` members are **strictly accessible only inside the class they are defined in** — they are not visible to any other class, including subclasses.

🎙️ Q4: WHAT'S THE DIFFERENCE BETWEEN `PROTECTED` AND `DEFAULT` ACCESS MODIFIERS? GIVE A SCENARIO WHERE ONE WORKS AND THE OTHER FAILS.

ANSWER:

SURE. THE MAIN DIFFERENCE BETWEEN `PROTECTED` AND `DEFAULT` ACCESS MODIFIERS IS HOW THEY BEHAVE ACROSS PACKAGES.

- **`default`** (no modifier) allows access **only within the same package** — regardless of whether the class is a subclass or not.
- **`protected`** also allows access within the same package — just like default — but it goes a step further: it allows access from a **subclass in a different package** as well, **as long as access happens through inheritance**.

💡 REAL-WORLD EXAMPLE:

LET'S SAY I HAVE A CLASS `BASEPAGE` IN `PACKAGEONE`, AND IT HAS A METHOD:

protected void waitForElement() { ... } //code

Then I have another class `LoginPage` in `packageTwo` that **extends `BasePage`**.
Since `waitForElement()` is `protected`, I can access it inside `LoginPage` using inheritance — even though they're in different packages.
But if the method was marked `default` (no modifier), the compiler would throw an error because **default doesn't allow cross-package access — not even in subclasses.**

🎙️ Q5: YOU'RE WRITING A TEST CASE IN A DIFFERENT PACKAGE AND WANT TO ACCESS A METHOD FROM ANOTHER CLASS. WHICH MODIFIER WILL ALLOW YOU TO DO THAT WITHOUT INHERITANCE?

ANSWER:

That would be `public`.
If I'm working in a different package and not using inheritance — meaning I'm accessing the method via an object — then the only modifier that allows that is `public`.
For example, in my SSR automation framework, I had utility classes like `DriverManager` in a base package. Methods like `getDriver()` or `closeDriver()` were marked `public`, so I could call them directly in test classes from other packages using:

```
DriverManager.getDriver();
```

🎯 WHY ACCESS MODIFIERS EXIST IN JAVA:

ACCESS MODIFIERS WERE CREATED TO CONTROL WHO CAN ACCESS WHAT — THEY HELP IN PROTECTING DATA,

ORGANIZING CODE, AND PREVENTING MISUSE.

💡 **HOW THEY MAKE LIFE EASIER:**

- `private` → Hides internal logic, so no one accidentally breaks it
- `public` → Lets you expose only what's necessary
- `protected` → Shares logic safely with child classes
- `default` → Keeps things limited to the same package, like a team boundary