Access Modifiers in Java

In Java, there are four main access modifiers that determine the visibility or accessibility of classes, methods, and variables. Here's an overview with code samples for each:

1. Public

- **Access**: Accessible from **anywhere** in the program.
- **Use**: When you want a class, method, or variable to be accessible by any other code.

```
// Public class, accessible from anywhere
public class PublicClass {
    // Public method, accessible from anywhere
    public void publicMethod() {
        System.out.println("This is a public method.");
    }
}
```

2. Protected

- **Access**: Accessible within the **same package** and by **subclasses** in other packages.
- **Use**: Primarily used in inheritance, allowing subclasses to use or override methods and variables while keeping them hidden from other classes.

```
public class ParentClass {
    // Protected variable, accessible within the same package
or in subclasses
    protected String protectedVar = "Protected Variable";

    // Protected method, accessible within the same package o
r in subclasses
```

```java
    protected void protectedMethod() {
        System.out.println("This is a protected method.");
    }
}

// Subclass in the same package
class ChildClass extends ParentClass {
    public void accessProtected() {
        System.out.println(protectedVar); // Accessible due to inheritance
        protectedMethod(); // Accessible due to inheritance
    }
}
```

3. Default (Package-Private)

- **Access**: Accessible **only within the same package** (no keyword needed).
- **Use**: When you want access to be limited to classes within the same package, providing a level of encapsulation without needing full protection.

```java
// Class with default access, accessible only within the same package
class DefaultClass {
    // Default variable, accessible only within the same package
    String defaultVar = "Default Variable";

    // Default method, accessible only within the same package
    void defaultMethod() {
        System.out.println("This is a default method.");
    }
}
```

4. Private

- **Access**: Accessible **only within the same class**.
- **Use**: To restrict access to methods or variables within the same class only, ensuring encapsulation and data hiding.

```
public class PrivateExample {
    // Private variable, accessible only within this class
    private int privateVar = 10;

    // Private method, accessible only within this class
    private void privateMethod() {
        System.out.println("This is a private method.");
    }

    // Public method that accesses private members
    public void accessPrivateMembers() {
        System.out.println("Private Variable: " + privateVa
r);
        privateMethod(); // Accessing private method within t
he same class
    }
}
```

Summary of Access Modifiers

Access Modifier	Class	Package	Subclass	World
`public`	✅	✅	✅	✅
`protected`	✅	✅	✅	❌
(default)	✅	✅	❌	❌
`private`	✅	❌	❌	❌

Example Usage in a Main Class

Here's a simple example to show how these modifiers impact accessibility:

```
public class Main {
    public static void main(String[] args) {
        PublicClass publicClass = new PublicClass();
        publicClass.publicMethod(); // Accessible

        ChildClass childClass = new ChildClass();
        childClass.accessProtected(); // Protected method acc
essed through inheritance

        DefaultClass defaultClass = new DefaultClass();
        // defaultClass.defaultMethod(); // Error: Not access
ible from different package

        PrivateExample privateExample = new PrivateExample();
        privateExample.accessPrivateMembers(); // Accesses pr
ivate members through a public method
    }
}
```

Recap

- **Public**: Use when you want everyone to access the code, regardless of location.
- **Protected**: Use when you want access within the same package and subclasses, useful for inheritance.
- **Default**: Use when you want to limit access to the same package only.
- **Private**: Use for encapsulation, keeping data and methods restricted to the same class.