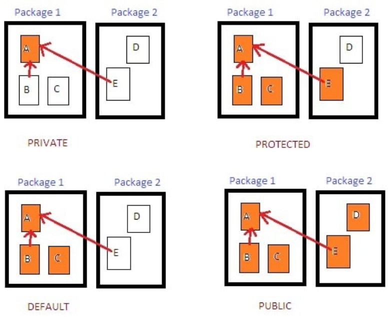
ACCESS MODIFIERS

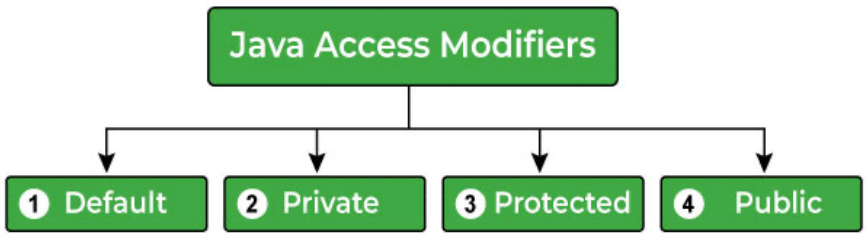
What are Access Modifiers?



in Java, Access modifiers help to restrict the scope of a class, constructor, variable, method, or data member. It provides security, accessibility, etc to the user depending upon the access modifier used with the element. Let us learn about Java Access Modifiers, their types, and the uses of access modifiers in this article.

Types of Access Modifiers

Primarily, there are four types of access modifiers. So, let's understand these different types of access modifiers in Java with examples.



Default Access Modifier

When no access modifier is specified for a class, method, or data member – It is said to be having the default access modifier by default. The data members, classes, or methods that are not declared using any access modifiers i.e. having default access modifiers are accessible only within the same package.

Private Access Modifier

The private access modifier is specified using the keyword **private**. The methods or data members declared as private are accessible only **within the class** in which they are declared.

Any other **class of** the **same package will not be able to access** these members. Top-level classes or interfaces cannot be declared as private because private means "only visible within the enclosing class".

Hence these modifiers in terms of application to classes, apply only to nested classes and not on top-level classes. **We cannot declare a top-level class as private**. If we try to do so, the compiler will throw an error. This is because top-level classes are meant to be either **public** (accessible from anywhere) or **default** (accessible within the same package). The private access modifier is only allowed for class members (fields, methods, constructors) and not for top-level classes.

Nested classes (i.e., inner classes) can be declared private. In this case, the private access modifier means that the nested class is only accessible within the enclosing class.

Inaccessible in Subclasses: A subclass **cannot access** a private method of its superclass, even though it inherits the class. Private methods are **not inherited** in the way public or protected methods are.

Singleton Pattern: A common use case for private constructors is to implement the **Singleton Pattern**. This pattern ensures that a class has only one instance and provides a global point of access to it.

Encapsulation: The private access modifier plays a crucial role in **encapsulation**, a fundamental principle of object-oriented programming (OOP). By marking fields as private and providing public getter and setter methods, you can hide the internal implementation details of a class and control how the data is accessed or modified.

Data Hiding: This helps in protecting the integrity of the object by preventing unauthorized external code from modifying its state. For example:

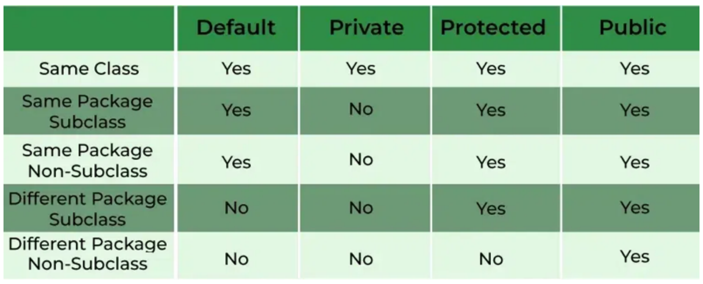
Protected Access Modifier

The protected access modifier is specified using the keyword **protected**.
The methods or data members declared as protected are **accessible within the same package or subclasses in different packages.**

Public Access Modifier

The public access modifier is specified using the keyword **public**.
The public access modifier has the **widest scope** among all other access modifiers.

Classes, methods, or data members that are declared as public are **accessible from everywhere** in the program. There is no restriction on the scope of public data members.



	Default	Private	Protected	Public
Same Class	Yes	Yes	Yes	Yes
Same Package Subclass	Yes	No	Yes	Yes
Same Package Non-Subclass	Yes	No	Yes	Yes
Different Package Subclass	No	No	Yes	Yes
Different Package Non-Subclass	No	No	No	Yes