En Java, los **modificadores de acceso** son palabras clave que establecen el nivel de visibilidad (o accesibilidad) para clases, atributos (variables), métodos y constructores. Son fundamentales para implementar el principio de **encapsulamiento**, uno de los pilares de la Programación Orientada a Objetos, que consiste en ocultar los detalles internos de una clase y exponer solo lo necesario.

## Tipos de Modificadores de Acceso

Java cuenta con cuatro tipos de modificadores de acceso, cada uno con un nivel de visibilidad diferente:

1. **public**
2. **protected**
3. **Default (sin modificador)**
4. **private**

Vamos a verlos en detalle, de más permisivo a más restrictivo:

### 1. public

El modificador **public** es el más abierto. Los elementos declarados como public son accesibles desde **cualquier lugar**.

* **Aplicable a:** Clases, atributos, métodos, constructores.
* **Visibilidad:**
  + Dentro de la misma clase.
  + Dentro del mismo paquete.
  + Desde subclases (incluso en otros paquetes).
  + Desde cualquier otra clase (incluso en otros paquetes).

**Ejemplo:**

Java

package com.miempresa.paquetea;  
  
public class ClasePublica { // La clase es pública  
 public int atributoPublico; // Atributo público  
  
 public ClasePublica(int valor) { // Constructor público  
 this.atributoPublico = valor;  
 }  
  
 public void metodoPublico() { // Método público  
 System.out.println("Este es un método público.");  
 }  
}

Java

package com.miempresa.paqueteb; // Otro paquete  
  
import com.miempresa.paquetea.ClasePublica;  
  
public class OtraClase {  
 public static void main(String[] args) {  
 ClasePublica obj = new ClasePublica(10); // Acceso al constructor público  
 System.out.println(obj.atributoPublico); // Acceso al atributo público  
 obj.metodoPublico(); // Acceso al método público  
 }  
}

**Uso:** Se usa para componentes de la API de una clase que deben ser visibles para todos los usuarios de la clase.

### 2. protected

El modificador **protected** permite el acceso dentro del mismo paquete y desde subclases, incluso si estas se encuentran en paquetes diferentes. Es ideal para elementos que deben ser accesibles por clases en la misma jerarquía de herencia.

* **Aplicable a:** Atributos, métodos, constructores. (¡No aplicable a clases de nivel superior!)
* **Visibilidad:**
  + Dentro de la misma clase.
  + Dentro del mismo paquete.
  + Desde subclases (incluso en otros paquetes).
  + **NO** desde cualquier otra clase fuera del paquete que no sea una subclase.

**Ejemplo:**

Java

package com.miempresa.paquetea;  
  
public class ClaseBase {  
 protected String mensajeProtegido = "Mensaje protegido de ClaseBase";  
  
 protected void metodoProtegido() {  
 System.out.println("Método protegido en ClaseBase.");  
 }  
}

Java

package com.miempresa.paqueteb; // Otro paquete  
  
import com.miempresa.paquetea.ClaseBase;  
  
public class SubclaseEnOtroPaquete extends ClaseBase { // Subclase en otro paquete  
 public void accederProtegido() {  
 System.out.println(this.mensajeProtegido); // Acceso permitido (subclase)  
 this.metodoProtegido(); // Acceso permitido (subclase)  
 }  
  
 public static void main(String[] args) {  
 SubclaseEnOtroPaquete objSub = new SubclaseEnOtroPaquete();  
 objSub.accederProtegido();  
  
 // ClaseBase objBase = new ClaseBase();  
 // System.out.println(objBase.mensajeProtegido); // ¡ERROR! No se puede acceder directamente desde fuera del paquete si no eres subclase.  
 }  
}

**Uso:** Común para atributos o métodos que las subclases necesitan modificar o extender, pero que no deberían ser parte de la API pública para cualquier otra clase.

### 3. Default (o "Package-Private")

Cuando no se especifica ningún modificador de acceso, se aplica el acceso **Default** (a veces llamado "package-private"). Los elementos con acceso default son accesibles solo desde clases dentro del **mismo paquete**.

* **Aplicable a:** Clases, atributos, métodos, constructores.
* **Visibilidad:**
  + Dentro de la misma clase.
  + Dentro del mismo paquete.
  + **NO** desde subclases en otros paquetes.
  + **NO** desde cualquier otra clase fuera del paquete.

**Ejemplo:**

Java

package com.miempresa.paquetea;  
  
class ClaseDefault { // La clase es default (sin modificador)  
 String atributoDefault = "Atributo default"; // Atributo default  
  
 void metodoDefault() { // Método default  
 System.out.println("Método default en ClaseDefault.");  
 }  
}

Java

package com.miempresa.paquetea; // Mismo paquete  
  
public class ClaseEnMismoPaquete {  
 public static void main(String[] args) {  
 ClaseDefault obj = new ClaseDefault(); // Acceso a la clase default  
 System.out.println(obj.atributoDefault); // Acceso al atributo default  
 obj.metodoDefault(); // Acceso al método default  
 }  
}

Java

package com.miempresa.paqueteb; // Otro paquete  
  
// import com.miempresa.paquetea.ClaseDefault; // ¡ERROR! No se puede importar una clase default de otro paquete.  
  
// public class ClaseEnOtroPaquete extends ClaseDefault { // ¡ERROR! No se puede heredar.  
// }

**Uso:** Para componentes que solo se deben usar internamente dentro de un paquete, ayudando a modularizar el código y a evitar dependencias no deseadas entre paquetes.

### 4. private

El modificador **private** es el más restrictivo. Los elementos declarados como private solo son accesibles desde **dentro de la propia clase** donde fueron definidos.

* **Aplicable a:** Atributos, métodos, constructores. (¡No aplicable a clases de nivel superior!)
* **Visibilidad:**
  + Dentro de la misma clase.
  + **NO** desde el mismo paquete.
  + **NO** desde subclases.
  + **NO** desde cualquier otra clase.

**Ejemplo:**

Java

package com.miempresa.paquetea;  
  
public class ClasePrivada {  
 private String datoSecreto = "ContraseñaUltraSecreta123"; // Atributo privado  
  
 private void metodoPrivadoInterno() { // Método privado  
 System.out.println("Acceso interno al método privado.");  
 }  
  
 public void accederDatosInternos() {  
 // La propia clase puede acceder a sus miembros privados  
 System.out.println("Dato secreto desde un método público: " + datoSecreto);  
 metodoPrivadoInterno();  
 }  
}

Java

package com.miempresa.paquetea;  
  
public class OtraClaseEnMismoPaquete {  
 public static void main(String[] args) {  
 ClasePrivada obj = new ClasePrivada();  
 obj.accederDatosInternos(); // Acceso permitido a través de un método público  
  
 // System.out.println(obj.datoSecreto); // ¡ERROR! No se puede acceder directamente  
 // obj.metodoPrivadoInterno(); // ¡ERROR! No se puede acceder directamente  
 }  
}

**Uso:** Es el pilar del **encapsulamiento**. Los atributos suelen ser private y se exponen a través de métodos public (getters y setters) para controlar cómo se accede o modifica su estado.

## Tabla Resumen de Visibilidad

| Modificador | Misma Clase | Mismo Paquete | Subclase (mismo paquete) | Subclase (otro paquete) | Otra Clase (mismo paquete) | Otra Clase (otro paquete) |
| --- | --- | --- | --- | --- | --- | --- |
| **public** | ✔️ | ✔️ | ✔️ | ✔️ | ✔️ | ✔️ |
| **protected** | ✔️ | ✔️ | ✔️ | ✔️ | ❌ | ❌ |
| **Default** | ✔️ | ✔️ | ❌ | ❌ | ✔️ | ❌ |
| **private** | ✔️ | ❌ | ❌ | ❌ | ❌ | ❌ |

## Consideraciones Adicionales:

* **Constructores Privados:** Un constructor private evita que la clase sea instanciada desde fuera de sí misma. Se usa comúnmente en clases singleton (que solo permiten una única instancia) o en clases de utilidades con solo métodos estáticos.
* **Clases Anidadas:** Las reglas de acceso tienen algunas particularidades con las clases anidadas (inner classes), donde la clase anidada tiene acceso a los miembros private de la clase envolvente.