Este post es un resumen de los modificadores de Java. Pensado para las personas que estén preparándose los exámenes de grado superior a modo de guía rápida.

Los modificadores de acceso permiten el encapsulamiento. El cual busca la forma controlar el acceso a los datos de un objeto o instancia.

Los modificadores dan seguridad a las aplicaciones limitando el acceso a diferentes atributos, métodos, constructores asegurando una "ruta" especificada acceder a la información.

Si nuestra aplicación es usada por programadores. Con modificadores de acceso aseguramos que un valor no será modificado incorrectamente. Para eso usaremos el acceso a los atributos con los métodos get y set.

Visibilidad	Public	Protected	Default	Private
Desde la misma Clase	SI	SI	SI	SI
Desde cualquier Clase del mismo Paquete	SI	SI	SI	NO
Desde una SubClase del mismo Paquete	SI	SI	SI	NO
Desde una SubClase fuera del mismo Paquete	SI	SI, a través de la herencia	NO	NO
Desde cualquier Clase fuera del Paquete	SI	NO	NO	NO

- Private

Este modificador solo nos permitirá acceder desde la misma clase. En caso de un método solo se podrá llamar internamente. Para los atributos, se crearan métodos públicos GETTERS y SETTERS.

```
package app.ejemplo;
public class Ejemplo1
```

```
{
        // Atributo y sus metodos
        private int atributo;

        public int getatributo()
        {
          return atributo;
        }

        public void setAtributo(int atributo)
        {
          this.atributo = atributo;
        }

        // Metodo
        private int metodo(){
          //code;
        }
}
```

- Default

Este acceso por defecto que permite que tanto la propia clase como las clases del mismo paquete.

```
package app.ejemplo;
public class Ejemplo2
{
    static int atributo = 0;
```

```
}
```

```
package app.ejemplo2;
public class Ejemplo2_1
{
    public static int getAtributo()
    {
            return Ejemplo2.atributo;
    }
}
```

- Protected

Nos permite acceder a las propiedades de la clase padre, cuando estemos haciendo herencias. Fuera de las clases hijas es equivalente a private.

```
package app.ejemplo;
public class Ejemplo3
{
    protected static int atributo = 0;
}
```

```
package app.ejemplo3_1;
import app.ejemplo3.Ejemplo3;
public class Ejemplo3_1 extends Ejemplo3
{
    public static void main(String[] args)
    {
        System.out.println(atributo)
```

```
    }
}
```

- Public

Es el mas permisivo. Cualquier clase tendra acceso a el sin importar paquete o procedencia.

```
package app.ejemplo4;
public class Ejemplo4
{
    public static int atributo = 0;

    public static void metodo()
    {
        System.out.println("Metodo Publico");
    }
}
```

```
package otro.paquete;
import app.ejemplo4.Ejemplo4;

public class OtraExterna
{
    public static void main(String[] args)
    {
        System.out.println(Ejemplo4.atributo);
        Ejemplo4.metodo();
    }
}
```

- La directiva Static

  Static es una directiva no un modificador de acceso, pero por utilidad vamos desarrollarla un poco.

  Una clase, método o campo declarado como estático puede ser accedido o invocado sin la necesidad de tener que instanciar un objeto de la clase. Uno de los ejemplos típicos de uso de métodos y variables estáticas es la clase java.lang.Math.

```
public class MathTest {
     public static void main(String[] args) {
        double floorOfPi = Math.floor(Math.PI);
        System.out.println(floorOfPi);
    }
}
```