Tabla de Contenidos


- Modificador private
- Modificador public
- Java modificadores acceso Protected
- Java Modificadores Acceso Protected y Herencia
- Conclusiones
- Otros Artículos relacionados

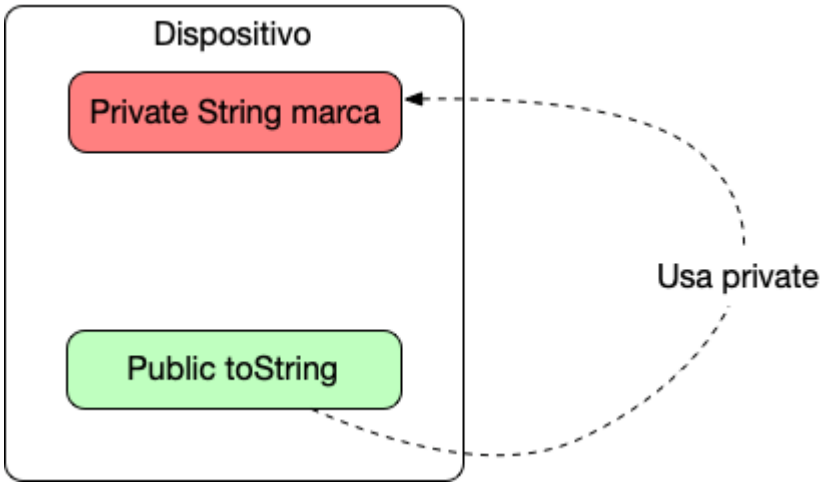

¿Java modificadores acceso y sus peculiaridades?. Cuando hablamos de modificadores de acceso en Java a todos nos vienen los mismos nombres “private” “public” “protected” . Son los modificadores de acceso que aportan un nivel de visibilidad a nuestros métodos y variables. En principio son muy sencillos de manejar y estamos acostumbrados a disponer de variables privadas y métodos públicos ya sean de funcionalidad o get/set. Veamos algo sencillo :

```
package com.arquitecturajava.dispositivos;

public class Dispositivo {

	private String marca;
	private double precio;
	public String getMarca() {
		return marca;
	}
	public void setMarca(String marca) {
		this.marca = marca;
	}
	public double getPrecio() {
		return precio;
	}
```

```
	public void setPrecio(double precio) {
		this.precio = precio;
	}
	public Dispositivo(String marca, double precio) {
		super();
		this.marca = marca;
		this.precio = precio;
	}
	@Override
	public String toString() {
		return "Dispositivo [marca=" + marca + ", precio=" +
precio + "]";
	}
}
```

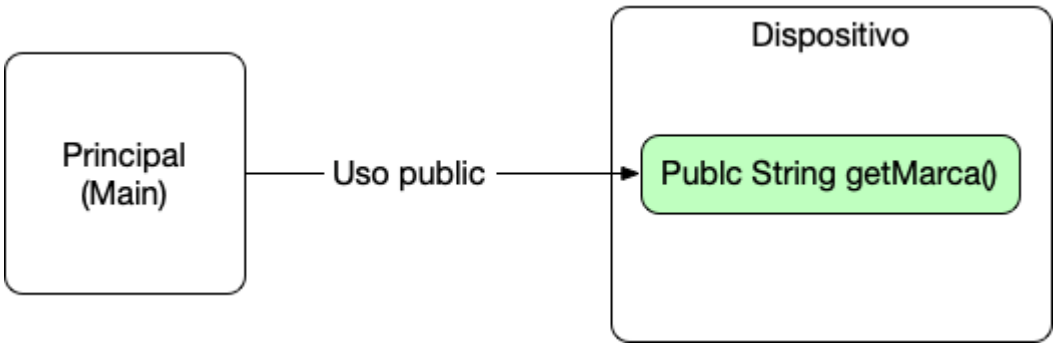
# Modificador private

Tenemos una clase Dispositivo y dispone de varias variables privadas a las cuales podemos acceder desde la propia clase como se muestra en el método toString(). Una variable de ámbito privado solo puede ser accesible desde la propia clase o desde Innerclases de esta esa es su visibilidad.



# Modificador public

Por otro lado tenemos el modificador public que hace referencia a cuando variable o método puede ser accedido desde cualquier lugar . En este caso podemos estar ante los métodos set/get o el constructor que se acceden desde un programa Principal cualquiera sin importar donde se encuentre este.




Lo vemos en código:

```
package com.arquitecturajava;

import com.arquitecturajava.dispositivos.Dispositivo;

public class Principal {

	public static void main(String[] args) {
		Dispositivo d= new Dispositivo("samsung",200);
		System.out.println(d.getMarca());
		System.out.println(d.getPrecio());

	}

}
```

Como se puede observar la clase Principal esta ubicada en otro package e importa la clase

dispositivo. No tiene ningún problema para acceder a los métodos get/set y constructor e invocarlos.

## Java modificadores acceso Protected

Este modificador cuesta entenderlo ya que un método protected le ve la clase actual en la que estamos y todas las clases hijas. Normalmente este modificador se usa para variables o funciones que nos pueden ser útiles en la propia clase como en clases hijas. Por ejemplo sencillo es que tuvieramos en la clase Dispositivo un método formatear que ayuda a formatear el método toString o cualquier otro.

```
package com.arquitecturajava.dispositivos;

public class Dispositivo {

	private String marca;
	private double precio;
	public String getMarca() {
		return marca;
	}
	public void setMarca(String marca) {
		this.marca = marca;
	}
	public double getPrecio() {
		return precio;
	}
	public void setPrecio(double precio) {
		this.precio = precio;
	}
	public Dispositivo(String marca, double precio) {
		super();
```

```
                this.marca = marca;
                this.precio = precio;
        }
        protected String formatear(String cadena) {
                return "|| "+ cadena+ "||";
        }
        @Override
        public String toString() {
                return  formatear("Dispositivo [marca=" + marca + ",
precio=" + precio + "]");
        }
}
```

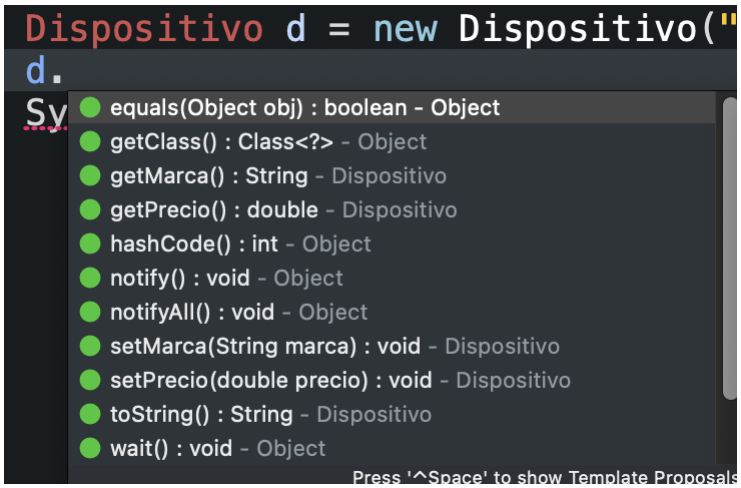
Si ahora imprimimos el objeto por la consola . Veremos sus datos un poco más formateados.

```
package com.arquitecturajava;

import com.arquitecturajava.dispositivos.Dispositivo;

public class Principal {

        public static void main(String[] args) {

                Dispositivo d = new Dispositivo("samsung", 200);
                System.out.println(d);

        }

}
```

```
|| Dispositivo [marca=samsung, precio=200.0]||
```

Acabamos de ver como usar un método protegido. ¿Qué tiene de peculiar? . Vamos a

intentar invocarlo desde la clase Principal que recordemos se encuentra en otro package que la clase Dispositivo.



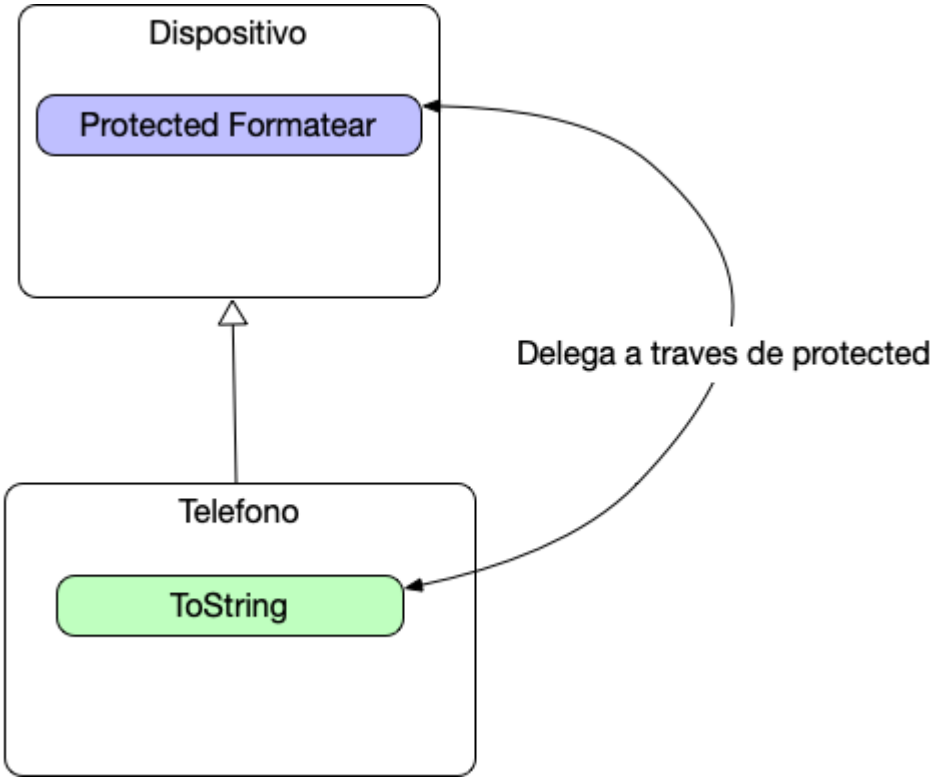

Como podemos ver no esta disponible. Cuando definimos un método protected este esta disponible para las clases que están en su mismo package y las clases hijas pero no esta disponible para ninguna clase adicional esto nos permite mantener una encapsulación fuerte.

# Java Modificadores Acceso Protected y Herencia

Al diseñar el método como protegido otras clases que hereden de la clase dispositivo pueden hacer uso de él sin ningún problema aunque estén en otro package. Aqui se muestra un ejemplo de la clase móvil.

```
package com.arquitecturajava.dispositivos.avanzandos;

import com.arquitecturajava.dispositivos.Dispositivo;

public class Movil  extends Dispositivo{

        private int numero;
```

```
	public Movil(String marca, double precio, int numero) {
		super(marca, precio);
		this.numero = numero;
	}

	public int getNumero() {
		return numero;
	}

	public void setNumero(int numero) {
		this.numero = numero;
	}

	@Override
	public String toString() {
		return formatear("numero:"+ numero);
	}

}
```




Nos queda invocar al método toString dentro de Movil y ver que todo compila perfectamente y se tiene acceso a él. Si probamos el método en un main.

```
package com.arquitecturajava;

import com.arquitecturajava.dispositivos.avanzandos.Movil;

public class Principal {

	public static void main(String[] args) {

		Movil m= new Movil("samsung",300,56789012);
		System.out.println(m);
	}

}
```

## Conclusiones

El resultado lo podemos ver en la consola con la sobrecarga del método toString. Acabamos de ver los diferentes modificadores acceso Java.

```
|| numero:56789012||
```

## Otros Artículos relacionados

- Java Herencia Multiple y sus opciones
- ¿Framework vs Frameworkless y cuál elegir?
- Funciones, Composición y Java
- ¿Que es un Java Trait?
- Acceso modificadores