Tabla de Contenidos

- Java Interface y métodos
- Java Interface Default Method
- Otros artículos relacionados

El concepto de Java Interface default method , es uno de los conceptos nuevos a nivel de Java 8 . Es verdad que estamos en Java 11 o Java 13 pero realmente los cambios importantes surgieron en Java 8 a nivel de programación funcional. ¿Para qué sirve un Java interface default method?. Vamos a explicarlo .

En primer lugar tenemos que decir que estos métodos son métodos que se implementan a nivel de interfaces . Esto siempre nos suena extraño ya que todos estamos mas que acostumbrados a que los interfaces solo definan la funcionalidad y no la implementen . Por lo tanto en muchas ocasiones la gente decide no usarlos y punto. Sin embargo no son tan difíciles de entender. Imaginemos que tenemos un conjunto de impresoras.

```
package com.arquitecturajava.ejemplo1;

public interface Impresora {

	public void imprimir(String texto);
	public int getVelocidad();
}
```

```
package com.arquitecturajava.ejemplo1;

public class ImpresoraLaser implements Impresora {
	private int velocidad;
	public ImpresoraLaser(int velocidad) {
		super();
		this.velocidad = velocidad;
```

```
        }
        public int getVelocidad() {
                return velocidad;
        }

        public void imprimir(String texto) {
                System.out.println("la impresora laser imprime :"+
texto);
        }

}
```

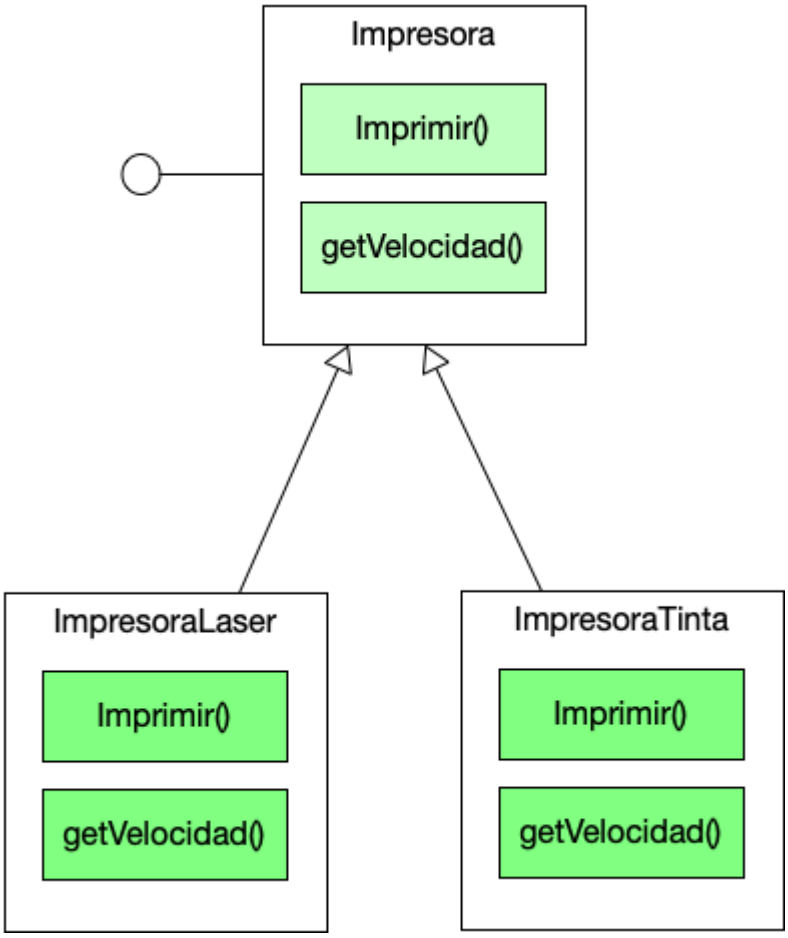
```
package com.arquitecturajava.ejemplo1;

public class ImpresoraTinta implements Impresora{

        private int velocidad;
        public ImpresoraTinta(int velocidad) {
                super();
                this.velocidad = velocidad;
        }
        public int getVelocidad() {
                return velocidad;
        }
        public void setVelocidad(int velocidad) {
                this.velocidad = velocidad;
        }
        public void imprimir(String texto) {
                System.out.println("la impresora de tinta imprime:" +
texto);
        }
```

```
}
```

En este caso lo que hemos definido es un interface Impresora con dos métodos imprimir() y getVelocidad() y hemos construido dos tipos de Impresora la Impresora Laser y la Impresora de Tinta. Hasta aquí todo correcto.



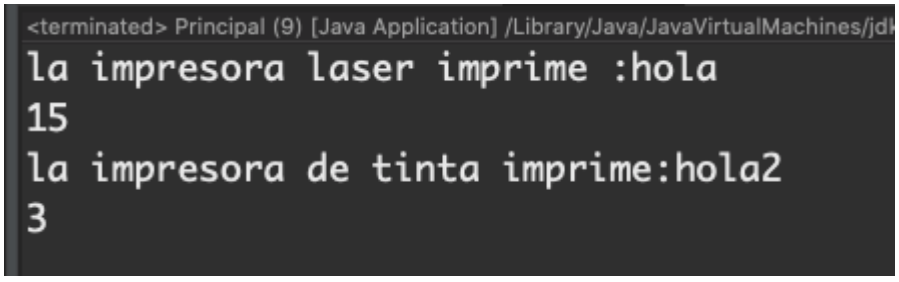
Java Interface y métodos

El interface impresora soporta los métodos imprimir (String texto) y getVelocidad() con los cuales imprimimos y obtenemos la velocidad de impresión de cada impresora a través de un programa main.

```
public class Principal {
```

```
	public static void main(String[] args) {
		Impresora i1= new ImpresoraLaser(15);
		Impresora i2= new ImpresoraTinta(3);
		i1.imprimir("hola");
		System.out.println(i1.getVelocidad());
		i2.imprimir("hola2");
		System.out.println(i2.getVelocidad());
	}

}
```



Hasta aquí es todo correcto pero mejorable . De hecho podemos darnos cuenta de que la propiedad de velocidad es un concepto compartido por ambas impresoras y lo podríamos subir a una clase abstracta que se llame TipoImpresora o algo similar, veamoslo:

```
package com.arquitecturajava.ejemplo2;

public abstract class TipoImpresora implements Impresora {

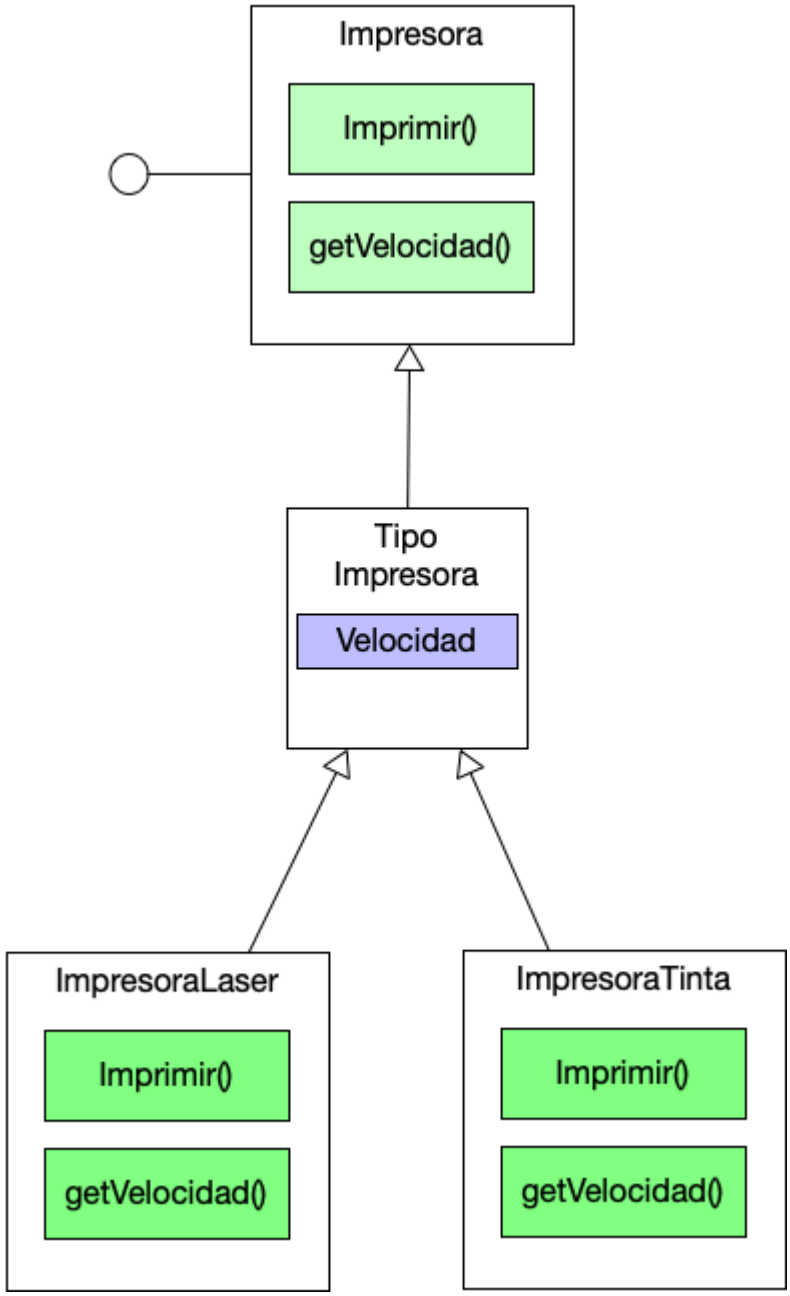
	private int velocidad;

	public TipoImpresora(int velocidad) {
		super();
		this.velocidad = velocidad;
	}
```

```
        public int getVelocidad() {
                return velocidad;
        }

        public void setVelocidad(int velocidad) {
                this.velocidad = velocidad;
        }
        public abstract void imprimir(String texto) ;
}
```

De esta forma el nuevo diagrama quedará algo diferente ya que incluirá una clase intermedia.

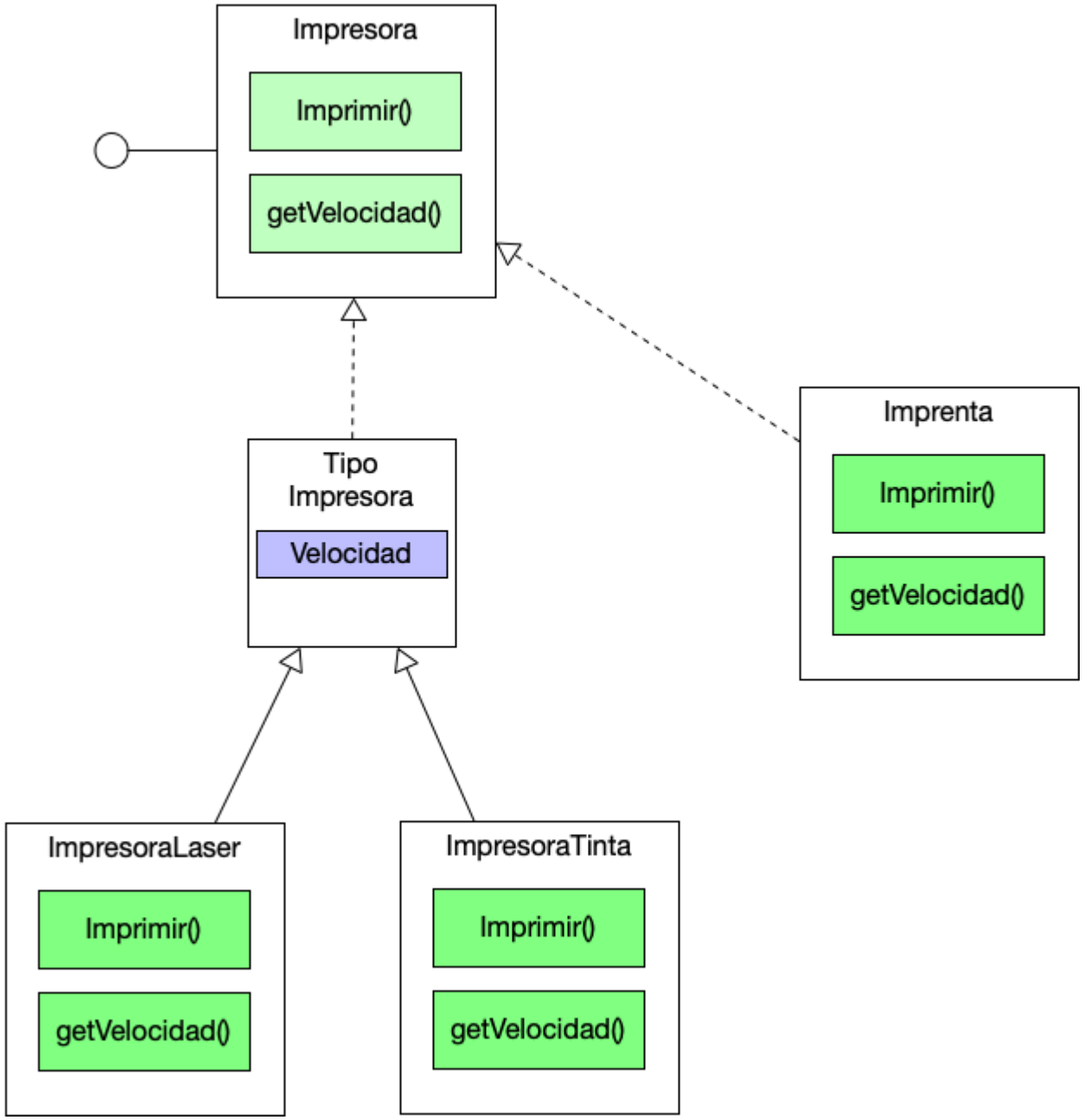


Las cosas quedan un poco mejor y es momento de ampliar la funcionalidad que tenemos ya que deseamos saber cuando una impresora es más rápida que otra. En principio el método parece trivial ya que nos valdría con construir algo como esto a nivel de la clase TipoImpresora:

```
public boolean esMasRapida(Impresora i) {
                if (i.getVelocidad()>this.getVelocidad()) {
                        return false;
                }else {
                        return true;
                }
        }
```

Java Interface Default Method

¿Es lo correcto? . La realidad es que depende mucho de lo que estemos buscando pero supongamos que aparece en la jerarquía de clases la clase Imprenta . Una Imprenta también puede imprimir documentos pero su velocidad es mucho más alta ya que usa un grupo de impresoras para imprimirlo todo . ¿Ahora bien es un TipoImpresora? . Si nos basamos en la regla de Liskov la realidad es que no lo es ya que por ejemplo una Imprenta nunca tendrá una propiedad marca o un peso. Por lo tanto el diagrama de clases será algo así:



El problema viene como entonces implementamos el método esMasRapida. La primera opción que teníamos pensada no nos es válida. Ya que no todas las clases que disponemos heredan de la clase TipoImpresora. La solución pasa por definir un Java Interface Default Methods en el interface Impresora.

```
public interface Impresora {
```

```
        public void imprimir(String texto);

        public int getVelocidad();

        public default boolean esMasRapida(Impresora i) {

                if (i.getVelocidad() > this.getVelocidad()) {
                        return false;
                } else {
                        return true;
                }
        }

}
```

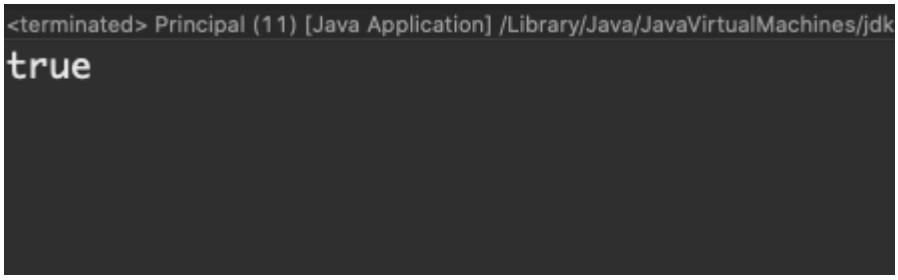
De esta manera el código será fuertemente reutilizable , podemos usar un programa main y probarlo:

```
package com.arquitecturajava.ejemplo3;

public class Principal {
        public static void main(String[] args) {
                Impresora i1= new ImpresoraLaser(15);
                Impresora i2= new ImpresoraTinta(3);
                Impresora i3= new Imprenta();
                System.out.println(i3.esMasRapida(i1));
        }

}
```

El resultado por la consola será true:



Acabamos de usar un Java Interface Default Method para mejorar el diseño de nuestro código:

Otros artículos relacionados

- Java 8 Stream
- Java Lambda
- Curso Java 8 Stream y Lamdas