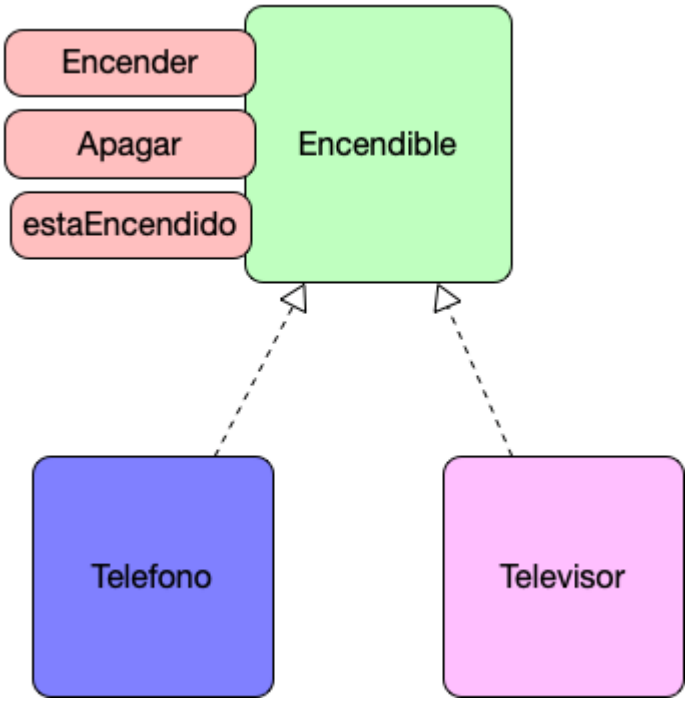
El concepto de Java Trait es uno de los conceptos más importantes de programación orientada a objeto. Eso sí en Java no estaban soportados hasta la llegada de Java 8 . ¿Cómo funciona un Java Trait? . En muchas ocasiones nos encontramos con clases que implementan un interface y que no tienen porque estar obligatoriamente ligadas en una jerarquía de clases. Vamos a ver un ejemplo con el interface Encendible que contiene los métodos encender , apagar y estaEncendido().

```
package com.arquiecturajava;

public interface Encendible {

	void encender();
	void apagar();
	boolean estaEncendido();
}
```

Este interface puede ser implementado por dos clases (Teléfono y Televisor) que no se encuentran en estos momentos ligados a ninguna jerarquía de clases.




```
package com.arquiecturajava;

public class Telefono implements Encendible {

	private boolean conectado;
	@Override
	public void encender() {
		conectado=true;
		System.out.println("el telefono se conecta");

	}

	@Override
	public void apagar() {
		conectado=false;
		System.out.println("el telefono se apaga");
```

```
        }

        @Override
        public boolean estaEncendido() {
                // TODO Auto-generated method stub
                return conectado;
        }

}
```

```
package com.arquiecturajava;

public class Televisor implements Encendible {

        private boolean encendido;

        @Override
        public void encender() {
                encendido = true;
                System.out.println(" el televisor se enciende");

        }

        @Override
        public void apagar() {

                encendido = false;
                System.out.println(" el televisor se apaga");

        }

        @Override
```

```
        public boolean estaEncendido() {
                // TODO Auto-generated method stub
                return encendido;
        }

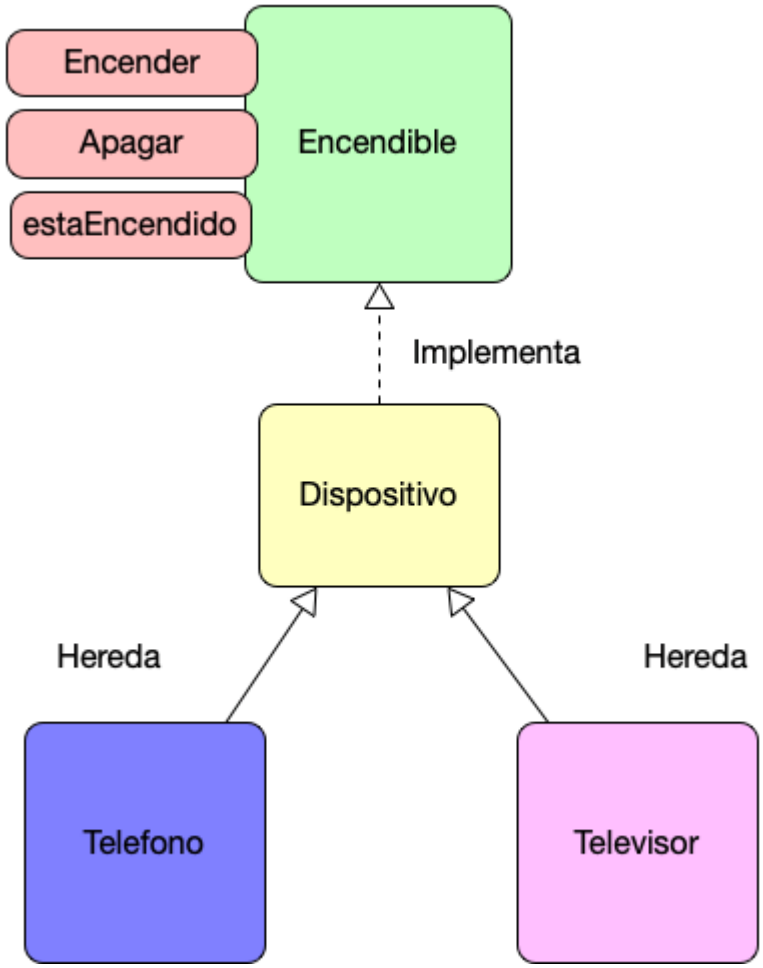
}
```

Clases Abstractas

Hasta aquí todo correcto . El problema comienza cuando tenemos algún método que a nivel de código es común para ambas clases. Por ejemplo en nuestro caso podría ser el método pulsarBoton que enciende el dispositivo si esta apagado y lo apaga si esta encendido.

```
 void pulsarBoton() {
                 if (estaEncendido()) {
                         apagar();
                 }else encender();
         }
```

Es un método sencillo pero no tenemos donde ubicarlo . Para poder ubicarlo no nos quedaría más remedio que construir la clase Abstracta Dispositivo y ubicarle allí .



Algo que quizás sea excesivo para la funcionalidad tan puntual que queremos definir junto con las dudas de si el interface encendible solo será capaz de encender dispositivos o podría encender una Casa también. Por lo tanto no queda claro como encajarlo.

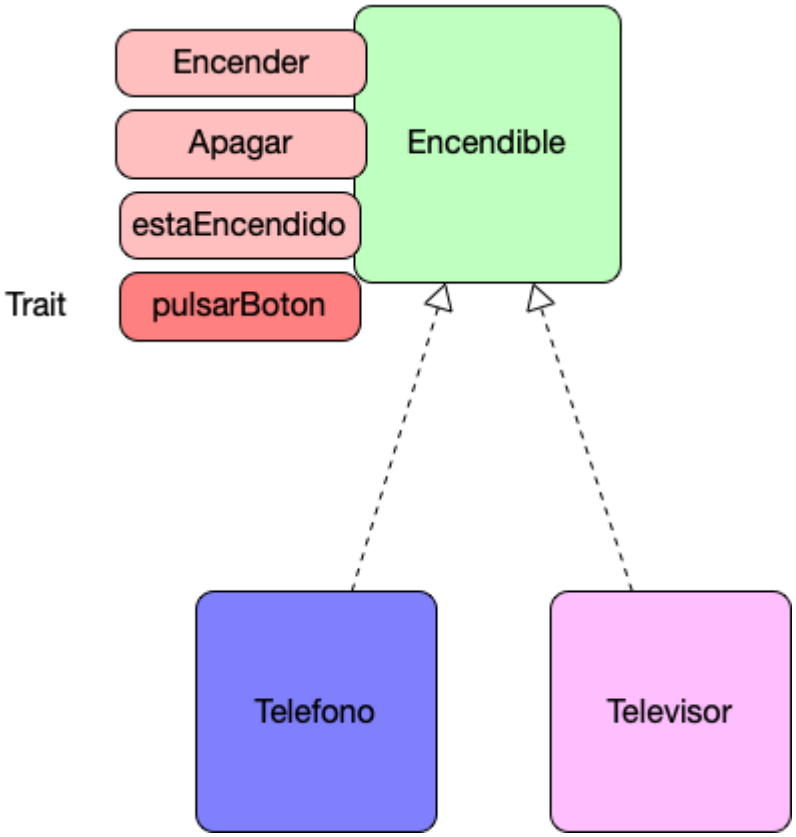
Java Trait y default methods

A partir de Java 8 , el lenguaje soporta default methods o métodos por defecto que permite añadir el método a la interface directamente sin tener que diseñar una clase abstracta intermedia . De esta forma el código quedaría:

```
package com.arquiecturajava;
```

```
public interface Encendible {

	void encender();
	void apagar();
	boolean estaEncendido();
	default  void pulsarBoton() {
		if (estaEncendido()) {
			apagar();
		}else encender();
	}
}
```



Es cuestión de construir un programa main y ver como tanto el Televisor como el Teléfono soportan de forma natural este método.

```
package com.arquiecturajava;

public class Principal {

	public static void main(String[] args) {
		Televisor t= new Televisor();
		t.pulsarBoton();
		Telefono t1= new Telefono();
		t1.pulsarBoton();
		t.pulsarBoton();
		t1.pulsarBoton();
	}

}
```

El resultado lo podemos ver en la consola:

```
 el televisor se enciende
el telefono se conecta
 el televisor se apaga
el telefono se apaga
```

Conclusiones

Acabamos de usar un Java Trait para simplificar el diseño de nuestras clases.

Otros artículos relacionados

- ¿Qué es un Java Lambda?
- ¿Que es un Java Stream?
- ¿Qué es un Java Optional?
- Default Methods