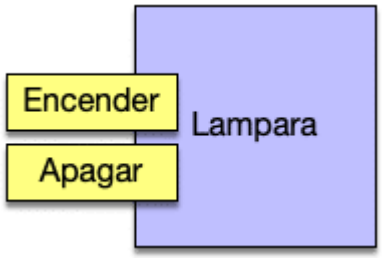
Java Adapter Pattern o patrón adaptador es uno de los patrones más utilizados hoy en día a nivel de desarrollo y probablemente es uno de los más sencillos de implementar una vez que hemos entendido como funciona. Ahora bien siempre resulta difícil de entender al principio. Vamos a construir un ejemplo que nos ayude a ver como usarle.

Imaginemonos que tenemos la clase Lampara . Esta clase es muy sencilla y dispone de una única propiedad “encendida” que nos devuelve si la lampara esta encendida o no.



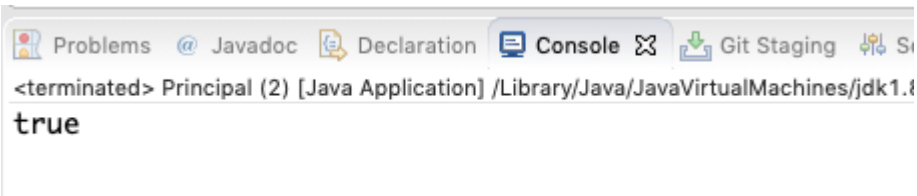

```
 package com.arquitecturajava;

public class Lampara {

        private boolean encendida;

        public boolean estaEncendida() {
                return encendida;
        }

        public void encender() {
                encendida=true;
        }
        public void apagar() {
                encendida=false;
        }
```

```
}
```

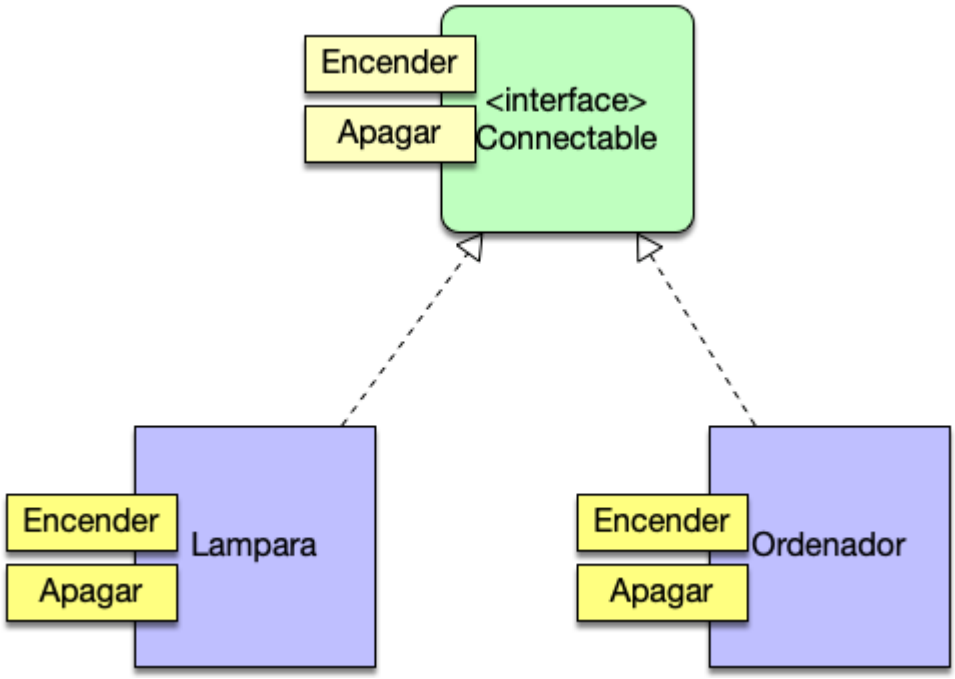
Podemos usar un programa principal para encender la lampara :

```
package com.arquitecturajava;

public class Principal {

	public static void main(String[] args) {
		Lampara l1= new Lampara();
		l1.encender();
		System.out.println(l1.estaEncendida());

	}

}
```

En este caso simplemente encendemos la lampara y comprobamos que esta encendida. El mensaje por la consola será “true”.



Hasta aqui todo correcto . En este momento mucha gente nos puede decir que claro que los métodos encender y apagar pertenecen mas a un interface que nos permita encender y apagar muchos tipos de objetos . Por ejemplo sería el caso también del ordenador. Esta claro que sí que ambos métodos quedarían mejor ubicados en un interface denominado Conectable o algo similar.



Esto hará que lo podamos usar de una forma más directa:

```
package com.arquitecturajava.ejemplo2;

public class Principal {

	public static void main(String[] args) {
		Conectable l1= new Lampara();
		l1.encender();
		System.out.println(l1.estaEncendida());
		Conectable o1= new Ordenador();
		o1.encender();
		System.out.println(o1.estaEncendida());
	}

}
```

Este código nos permite gestionar tanto el Ordenador como la Lampara como si fueran tipos

Conectables. Eso sí sería mucho más compacto el código de la siguiente forma.

```
package com.arquitecturajava.ejemplo2;

public class Principal {

	public static void main(String[] args) {
		Conectable l1= new Lampara();
		encenderAparato(l1);
		Conectable o1= new Ordenador();
		encenderAparato(o1);

	}

	private static void encenderAparato(Conectable l1) {
		l1.encender();
		System.out.println(l1.estaEncendida());
	}

}
```

Hemos usado una función para no repetir tanto código . Esta recibe un objeto de tipo Conectable y enciende el elemento que se solicite. El problema viene cuando queremos que este código funcione con una clase como esta:

```
package com.arquitecturajava.ejemplo2;

public class LamparaInglesa {

	private boolean isOn;
	public boolean isOn() {
		return isOn;
```

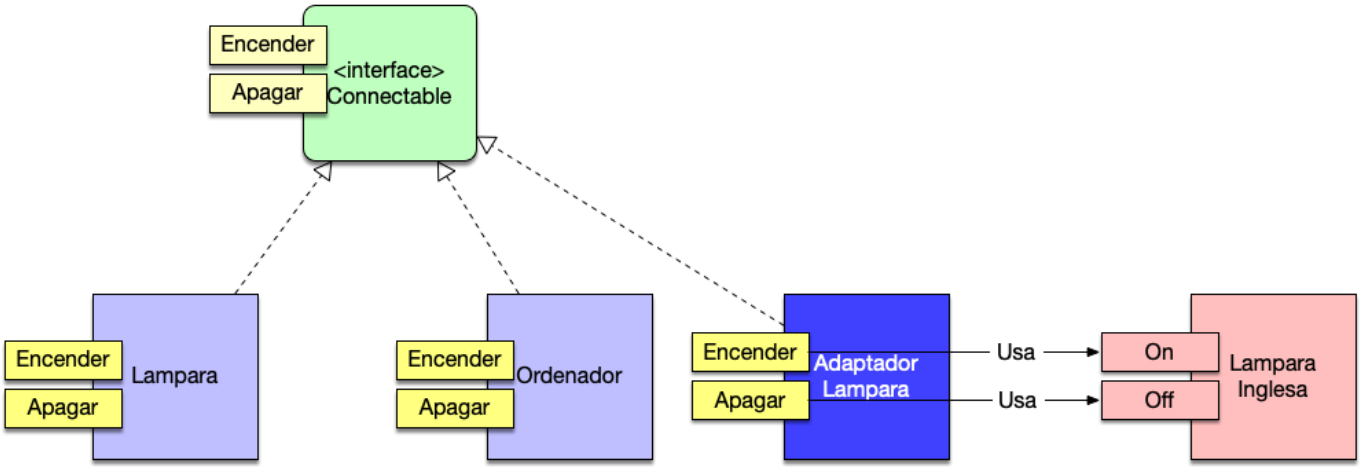
```
        }

        public void on () {
                isOn=true;
        }
        public void off() {
                isOn=false;
        }
}
```

Lamentablemente nuestra Lampara inglesa es un código creado por un programador ajeno a la empresa y construyo lo que le dio la gana. Así son las cosas.

Supongamos que solo disponemos del fichero JAR y no del código fuente o por lo que sea no queremos cambiarlo . Tenemos un problema importante ya que no le podemos integrar dentro de nuestro programa e interface Conectable. ¿Cómo podemos solventar algo de este estilo?.

Java Adapter Pattern , la solución

Para solventar este problema podemos usar Java Adapter Pattern , o patrón adaptador que se encargará de construir una clase que adapte la funcionalidad de la clase LamparaInglesa de tal forma que se pueda usar dentro de nuestra estructura de conectables.



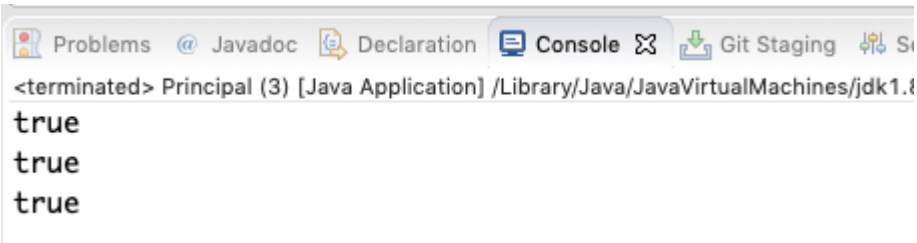
Vamos a ver el código de la clase y ver como esta clase “AdaptadorLampara” implementa el interface Conectable delegando en la lampara inglesa

```
package com.arquitecturajava.ejemplo2;

public class AdaptadorLampara implements Conectable{

	private LamparaInglesa lampara= new LamparaInglesa();

	public boolean estaEncendida() {
		return lampara.isOn();
	}

	public void encender() {
		lampara.on();
	}

	public void apagar() {
		lampara.off();
	}
}
```

Ahora bastaría con modificar el programa principal.

```
package com.arquitecturajava.ejemplo2;

public class Principal {

	public static void main(String[] args) {
		Conectable l1= new Lampara();
		encenderAparato(l1);
		Conectable o1= new Ordenador();
		encenderAparato(o1);
		Conectable l2= new AdaptadorLampara();
		encenderAparato(l2);

	}

	private static void encenderAparato(Conectable l1) {
		l1.encender();
		System.out.println(l1.estaEncendida());
	}

}
```

Ahora si saldrán los tres como encendidos:



El uso de Java Adapter pattern nos puede ayudar a solventar muchas situaciones de

programación compleja en donde tenemos que integrar código de distintos desarrolladores.

- Java Command Pattern
- Java Factory
- Inyección de Dependencia
- WikiPedia