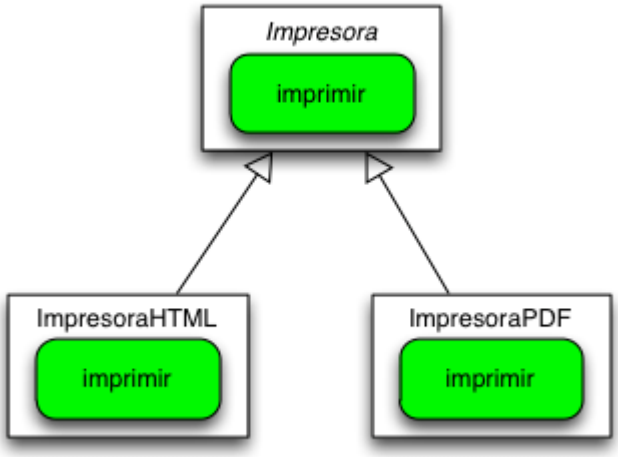
Trabajar con patrones de diseño nunca es sencillo y siempre resulta complejo ver como se relacionan con unos u otros principios de Ingeniería. Hoy voy a mostrar un ejemplo del patrón adaptador. Supongamos que tenemos la siguiente jerarquia de clases que definen varias impresoras que imprimen documentos en distintos formatos.



Vamos a ver el código fuente de estas clases (Impresora)

```
public abstract class Impresora {

private String texto;

public String getTexto() {
 return texto;
 }

public void setTexto(String texto) {
 this.texto = texto;
```

```
 }

 public abstract void imprimir();
}
```

(Impresora Texto)

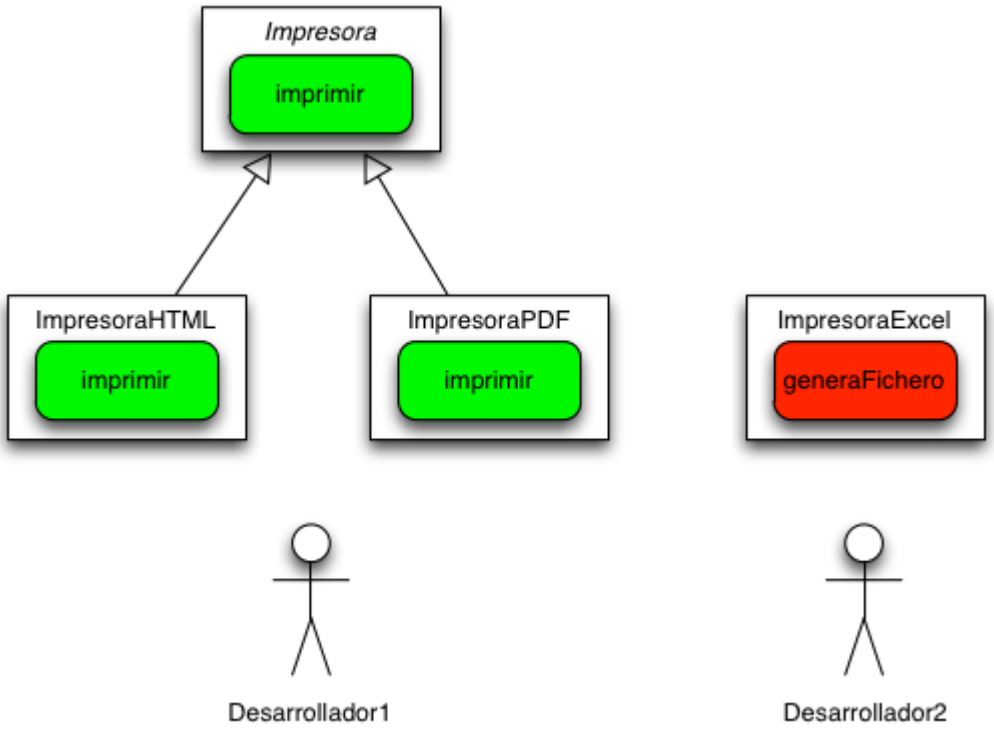
```
public class ImpresoraTexto extends Impresora {

 public ImpresoraTexto(String texto) {
 super();

}

@Override
 public void imprimir() {

 System.out.println("fichero texto con "+ getTexto());

}
}
```

(Impresora PDF)

```
public class ImpresoraPDF extends Impresora {
```

```
public ImpresoraPDF(String texto) {
 super();
 setTexto(texto);
 }
@Override
 public void imprimir() {

System.out.println("fichero pdf con " + getTexto());

}

}
```

Hemos creado una jerarquia de clases en la que el método imprimir (pseudo código) se encargaría de generar los distintos tipos de documentos .Ahora bién que pasaría si otro desarrollador ya hubiera desarrollado una clase para ficheros excel y quisieramos usarla dentro de nuestra jerarquia.



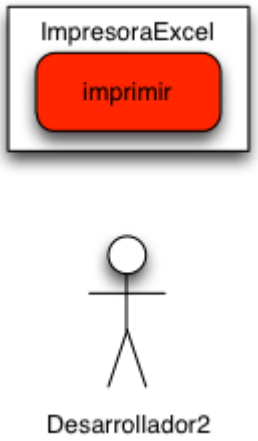
Vamos a ver su código fuente

```
public class ImpresoraExcel {
 public void generarFichero(String texto) {

System.out.println("fichero excel con " + texto);

}
}
```

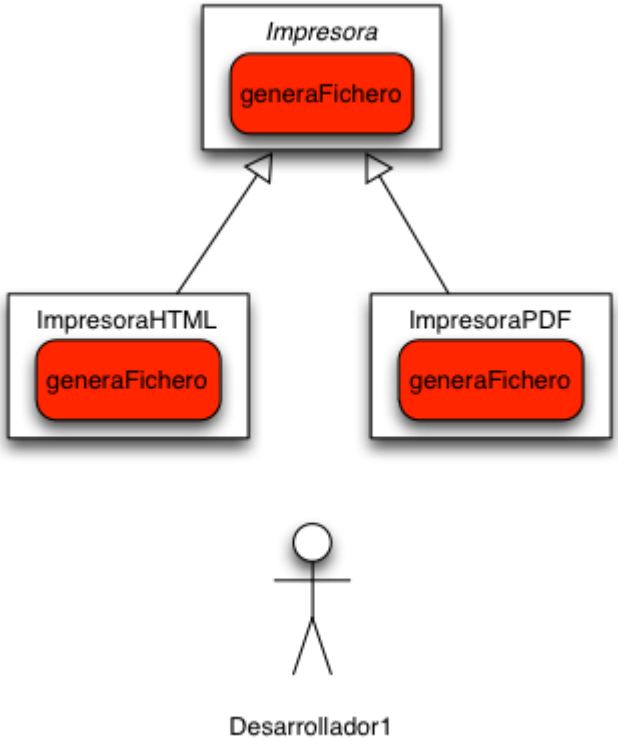
Nos podemos dar cuenta de que la clase no encaja en nuestra jerarquía actual ya que no

hereda de la clase impresora y no comparte el método imprimir. Existen dos caminos erroneos básicos que se suelen tomar en estos casos.

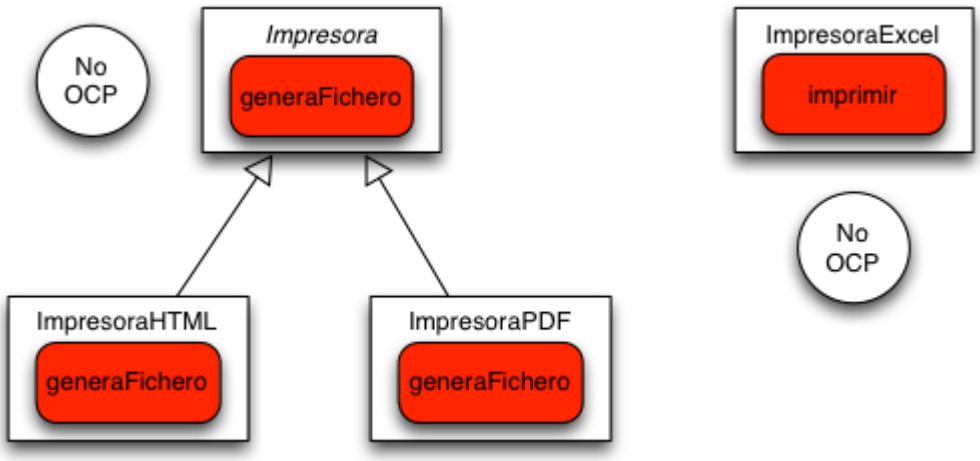
1. Modificar la clase de Impresora para que encaje en la jerarquia de clases



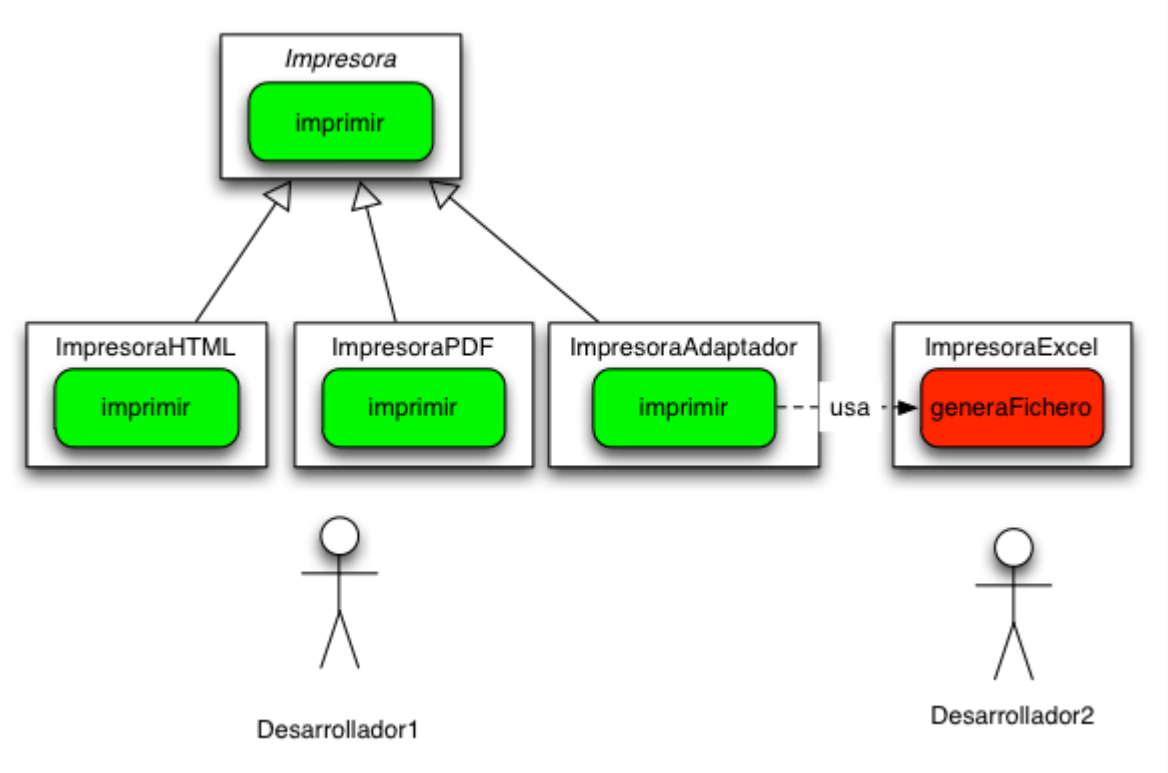
2. Modificar nuestra jerarquia de clases para que encaje con la clase ImpresoraExcel lo cual supone todavía un mayor esfuerzo.



Sin embargo ambas soluciones rompen con el principio OCP (Una clase debe estar abierta a la extensibilidad pero cerrada a las modificaciones) .



Si queremos encajar de una manera correcta la clase ImpresoraExcel en nuestra jerarquia de clases deberemos hacer uso de una clase Adaptador . Esta clase hereda de la clase impresora y se encarga de adaptar los métodos de la clase ImpresoraExcel a la jerarquía que ya poseemos.



A continuación se muestra el código fuente de esta clase:

```
public class ImpresoraAdaptador extends Impresora {

private ImpresoraExcel impresoraExcel;

 public ImpresoraAdaptador() {
```

```
 super();
 impresoraExcel= new ImpresoraExcel();
 }

@Override
 public void imprimir() {

 impresoraExcel.generarFichero(getTexto());

 }

}
```

De esta forma conseguimos cumplir con el principio OCP y no tener que modificar el código previamente construido.Viendo la relación existente entre los patrones de diseño y los principios de ingeniería.