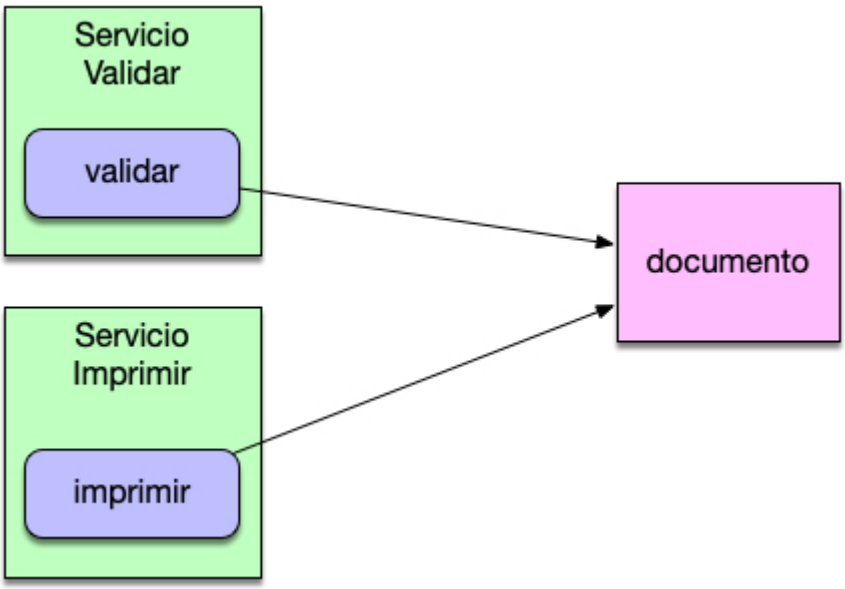
La pregunta de Adapter vs Facade a nivel de design pattern es una de las preguntas más habituales cuando uno empieza a trabajar con ambos patrones y a veces su similitud hace que no entendamos de forma correcta como se usa cada uno de estos patrones. Vamos a echar un vistazo a ambos y explicar sus diferencias . Para ello el primer paso es construir dos clases de Servicio que aporten una funcionalidad elemental a la clase Documento. La primera clase se encarga de validar el documento y la segunda de imprimirlo por la consola , es código sencillo.



Veamos el código de estas tres clases :

```
package com.arquitecturajava;

public class Documento {

  private String texto;
  private String firma;
  public String getTexto() {
    return texto;
  }
```

```
  public void setTexto(String texto) {
    this.texto = texto;
  }
  public String getFirma() {
    return firma;
  }
  public void setFirma(String firma) {
    this.firma = firma;
  }
}
```

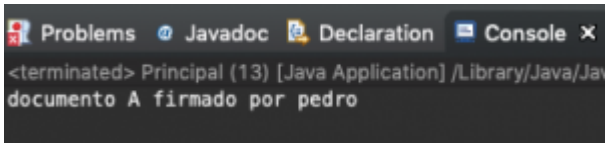
```
package com.arquitecturajava;

public class ServicioImprimirDocumento {

  public void imprimir(Documento c) {
    System.out.println(c.getTexto());
  }
}
```

```
package com.arquitecturajava;

public class ServicioValidarDocumento {

  public boolean validar(Documento documento) {
    if (documento.getTexto().contains(documento.getFirma())) {
      return true;
    }else {
      return false;
    }
  }
}
```

En este caso estamos viendo que la clase ServicioImprimir documento simplemente imprime el contenido del documento por la consola . Mientras que la clase ServicioValidarDocumento se encarga de confirmar que la firma que pasamos al documento esta incluida en el propio texto del documento . De esta forma podremos dar por válido el documento que se construye. Vamos a construir un programa que use ambos servicios.

```
package com.arquitecturajava;

public class Principal {

  public static void main(String[] args) {
    Documento documento= new Documento("documento A firmado por
pedro","pedro");
    ServicioValidarDocumento servicioValidarDocumento= new
ServicioValidarDocumento();
    if(servicioValidarDocumento.validar(documento)) {
      ServicioImprimirDocumento servicioDocumento= new
ServicioImprimirDocumento();
      servicioDocumento.imprimir(documento);
    }
  }

}
```

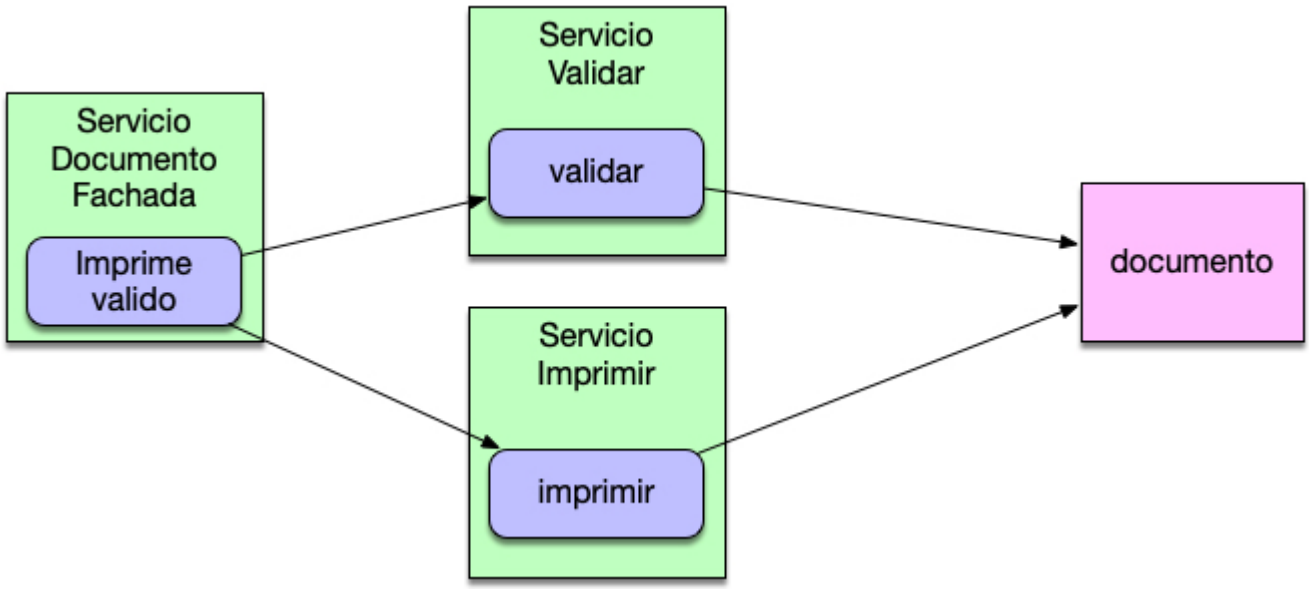
Si ejecutamos el programa main nos devolverá el siguiente resultado :



Todo funciona correctamente

Adapter vs Facade (El patrón fachada)

El patrón fachada o Facade aparece para solventar el problema de que en muchos casos existe una funcionalidad que deseamos ejecutar de forma reiterativa y que afecta a varios servicios por ejemplo en este caso podría hacer referencia a que primero debemos validar un documento antes de de poderlo imprimir . Esta funcionalidad se construye a través del uso combinado de dos servicios. Por lo tanto podemos construir un nuevo Servicio que aglutine la funcionalidad de los dos anteriores y le podemos denominar ServicioDocumentoFachada como vemos en el siguiente diagrama.



Veamos su código:

```
package com.arquitecturajava;

public class ServicioDocumentoFachada {
  private ServicioImprimirDocumento servicioImprimir;
  private ServicioValidarDocumento servicioValidar;
```

```
  public ServicioDocumentoFachada() {
    this.servicioImprimir=new ServicioImprimirDocumento();
    this.servicioValidar=new ServicioValidarDocumento();
  }
  public void ImprimeValido(Documento d) {
    if (servicioValidar.validar(d)) {
      servicioImprimir.imprimir(d);
    }
  }

}
```

De esta manera a la hora de construir el programa Principal todo es mucho más sencillo basta con construir el nuevo servicio y este delegará en los otros:

```
package com.arquitecturajava;

public class Principal2 {

  public static void main(String[] args) {
    Documento documento= new Documento("documento A firmado por
pedro","pedro");
    ServicioDocumentoFachada fachada= new ServicioDocumentoFachada();
    fachada.ImprimeValido(documento);
  }

}
```

Las fachadas de construyen para simplificar y ayudar a aumentar la reutilización cuando combinamos varios servicios de forma reiterativa.

Adapter vs Facade (El patrón Adapter)

¿Para qué sirve entonces el patrón adaptador o adapter? . La casuística puede resultarnos similar ya que nos encontramos ante una situación en la cual nosotros volvemos a hacer uso del concepto de delegación. Los adaptadores como su nombre indica sirven para “adaptar” . Donde puede encajar un adaptador dentro de nuestro ejemplo . Imaginemos que estamos contentos con nuestra fachada que se encarga de validar documentos. Sin embargo alguien que pertenece al equipo de desarrollo nos dice que tiene un problema con nuestra solución y es que él ya tiene construida una clase Documento que ha usado en muchos lugares.

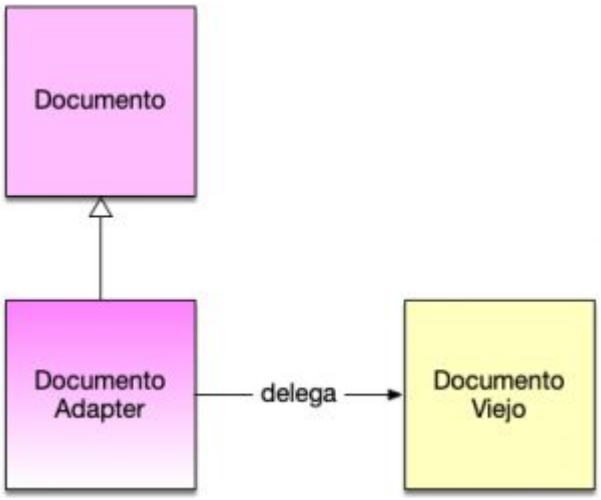
Veamos su código:

```
package com.arquitecturajava;

public class DocumentoViejo {

  private String contenido;
  private String  firmaValida;
  public String getContenido() {
    return contenido;
  }


  public void setContenido(String contenido) {
    this.contenido = contenido;
  }


  public String getFirmaValida() {
    return firmaValida;
  }
```

```
  public void setFirmaValida(String firmaValida) {
    this.firmaValida = firmaValida;
  }


  public DocumentoViejo(String contenido, String firmaValida) {
    super();
    this.contenido = contenido;
    this.firmaValida = firmaValida;
  }
}
```

Si nos fijamos la clase es prácticamente idéntica simplemente cambian los nombres. Rapidamente le diremos a nuestro programador que no se preocupe que nuestra clase es muy muy similar y que puede usarla sin ningún tipo de problemas. El problema es que el nos responderá que no quiere usar esta clase ya que tiene muchos de sus Servicios que ya devuelven o reciben DocumentoViejo como tipo y por lo tanto quiere que nosotros modifiquemos nuestra clase de servicioFachada para que admita DocumentoViejo como concepto.

Nuestra respuesta será parecido a la suya y es que no queremos modificar nuestra funcionalidad ya que tenemos mucho código construido. ¿Qué podemos hacer para encajar las soluciones de las dos personas al mismo tiempo sin asumir grandes cambios? . Muy sencillo podemos construir un adaptador para la clase DocumentoViejo que haga que esta clase se pueda encajar dentro del concepto de Documento en el cual estamos trabajando con nuestros servicios. En este caso vamos a usar una combinación de herencia y delegación para construir esta clase.



Lo que estamos haciendo es extender la clase Documento y generar una clase que sea "parecida" y que permite adaptar el concepto de DocumentoViejo para que encaje en nuestra solución.

Veamos el código de esta clase:

```
package com.arquitecturajava;

public class DocumentoAdapter extends Documento {

  private DocumentoViejo documentoViejo;

  public DocumentoAdapter(DocumentoViejo documento) {

    this.documentoViejo = documento;

  }

  @Override
  public String getTexto() {
```

```
    return documentoViejo.getContenido();
  }

  @Override
  public void setTexto(String texto) {
    documentoViejo.setContenido(texto);
  }

  @Override
  public String getFirma() {
    return documentoViejo.getFirmaValida();
  }

  @Override
  public void setFirma(String firma) {
    documentoViejo.setFirmaValida(firma);
  }

}
```

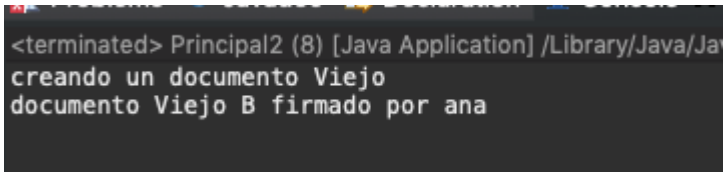
Al usar el Adaptador hemos adaptado los métodos de la clase DocumentoViejo a los métodos que necesita la clase Documento y que por lo tanto son los que necesita nuestro servicio. Vamos a añadir un mensaje de traza al constructor de la clase DocumentoViejo y ver este código en acción

```
public DocumentoViejo(String contenido, String firmaValida) {
  System.out.println("creando un documento Viejo");
  this.contenido = contenido;
  this.firmaValida = firmaValida;
}
```

Es momento de probar este código con el adaptador en un programa Principal

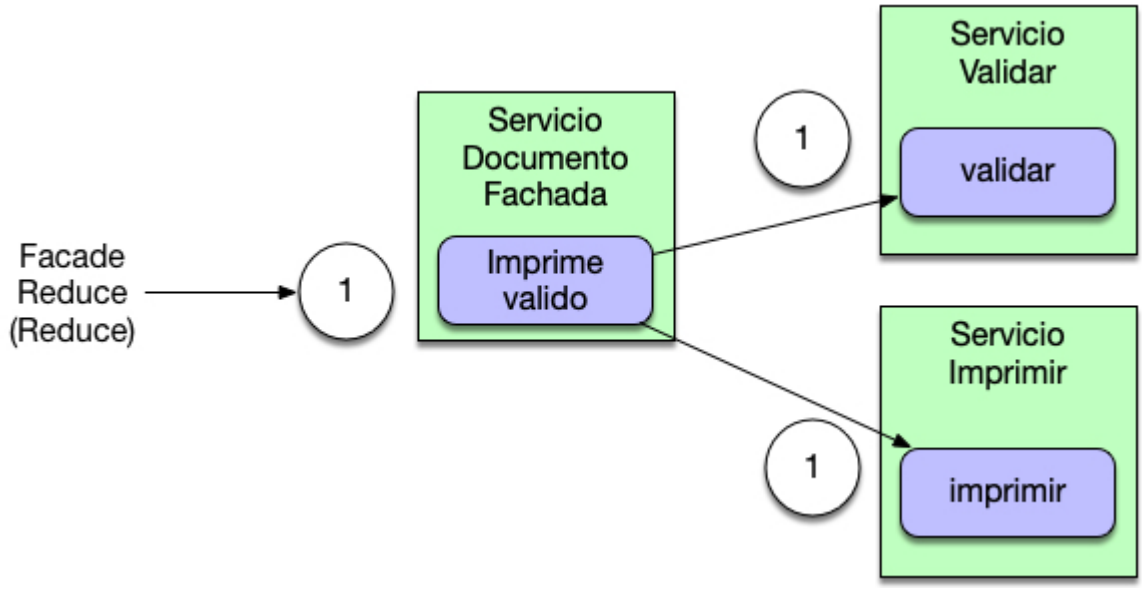
```
package com.arquitecturajava;

public class Principal2 {

  public static void main(String[] args) {
    DocumentoViejo documentoViejo= new DocumentoViejo("documento Viejo
B firmado por ana","ana");
    ServicioDocumentoFachada fachada= new ServicioDocumentoFachada();
    fachada.ImprimeValido(new DocumentoAdapter(documentoViejo));
  }

}
```

Podemos ver perfectamente como el adaptador es capaz de hacer encajar un documento viejo dentro del servicio que acabamos de construir. Si ejecutamos el código nos daremos cuenta que se construye el objeto viejo y que fue firmado sin ningún tipo de problemas.

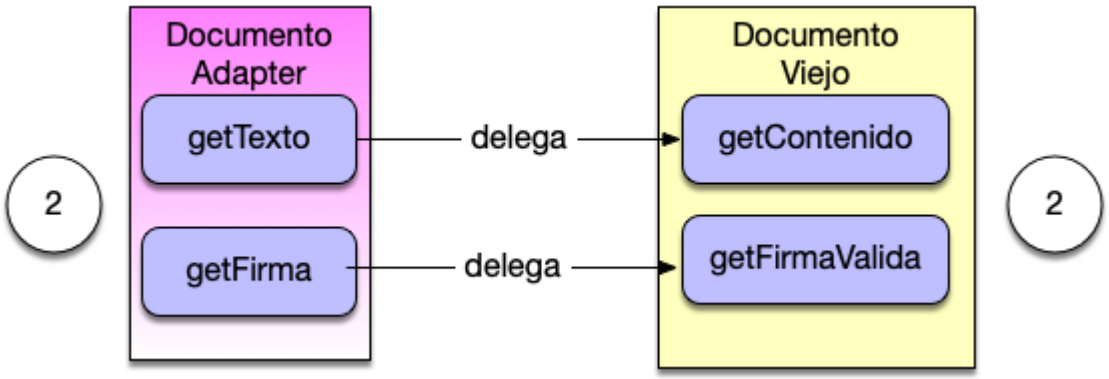


Delegación y patrones

¿Porqué es tan habitual confundir el patrón Facade con el patrón Adaptador? . Es muy común porque ambos patrones usan el concepto de delegación para trabajar con las clases . Sin embargo existe una diferencia fundamental el patrón fachada o facade usa la delegación y reduce el número de métodos a utilizar . Es decir en nuestro caso ServicioValidar dispone de un método y ServicioImprimir dispone de otro , total de métodos 2 . En cambio la Fachada solo dispone de un único método ya que agrupa los otros métodos en uno solo.



¿Qué ocurre en el caso del adaptador? . En el caso del adaptador se mantienen los métodos que ambas clases tienen ya que se usa la delegación pero sin agrupar métodos y simplemente cambiar su nombre o los parámetros que recibe.



Esta es la diferencia que existe entre Adapter vs Facade

Otros artículos relacionados

1. Adaptadores y patrones y el principio OCP

2. Command Pattern en Java y la gestion de tareas
3. Java 8 Factory Pattern y su implementación
4. Patrones