El concepto de acoplamiento es uno de los conceptos que más cuesta entender cuando trabajamos en nuestro día a día y hace referencia a como nuestras clases están relacionados entre ellas y las dependencias que tienen. Vamos a construir un ejemplo sencillo que nos permita entenderlo mejor. Para ello nos vamos a construir una serie de servicios que se encarguen de manipular la clase Factura.

```
package com.arquitecturajava;

public class Factura {

  private int numero;
  private String concepto;
  private double importe;
  public int getNumero() {
    return numero;
  }
  public void setNumero(int numero) {
    this.numero = numero;
  }
  public String getConcepto() {
    return concepto;
  }
  public void setConcepto(String concepto) {
    this.concepto = concepto;
  }
  public double getImporte() {
    return importe;
  }
  public void setImporte(double importe) {
    this.importe = importe;
  }
```

```
  public Factura(int numero, String concepto, double importe) {
    super();
    this.numero = numero;
    this.concepto = concepto;
    this.importe = importe;
  }
}
```

Una vez tenemos la Factura construida es momento de construir una serie de clases que hagan operaciones sencillas con ellas.

```
package com.arquitecturajava.servicios;

import com.arquitecturajava.Factura;

public class ServicioValidacion {

  public boolean validarFactura(Factura f) {
    if (f.getImporte()<100) {
      System.out.println("factura valida");
      return true;
    }else {
      return false;
    }
  }
}
```

```
package com.arquitecturajava.servicios;

import com.arquitecturajava.Factura;

public class ServicioPago {
```

```
  public void pagarFactura(Factura f, String cuenta) {
    System.out.println("pago en cuenta:"+ cuenta);
    System.out.println("importe a pagar"+ f.getImporte());
  }
}

}
```

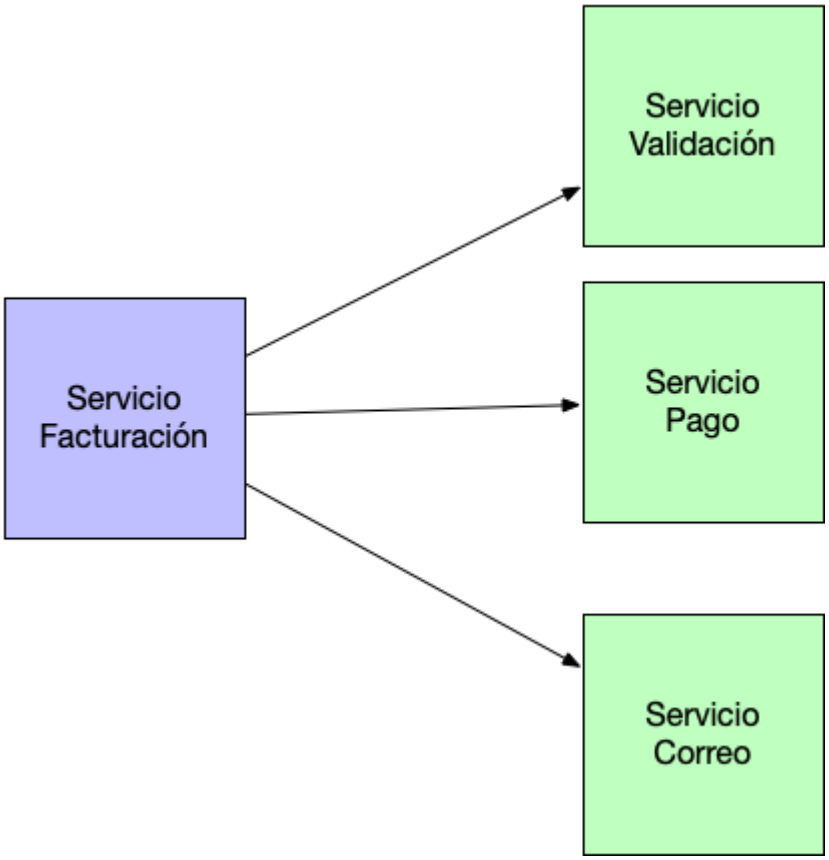
```
package com.arquitecturajava.servicios;

public class ServicioCorreo {
  public void enviarMensaje(String email,String mensaje) {
    System.out.println("enviando correo a :"+ email);
    System.out.println("el mensaje es:"+ mensaje);
  }

}
```

Una vez tenemos estos tres servicios funcionando es momento de crear un nuevo servicio que procese la factura delegando en el resto. Una opción razonable es :

```
package com.arquitecturajava.servicios;

import com.arquitecturajava.Factura;

public class ServicioFacturacion {

  private ServicioValidacion servicioValidacion;
  private ServicioCorreo servicioCorreo;
  private ServicioPago servicioPago;
  public ServicioFacturacion(ServicioValidacion servicioValidacion,
ServicioCorreo servicioCorreo,
```

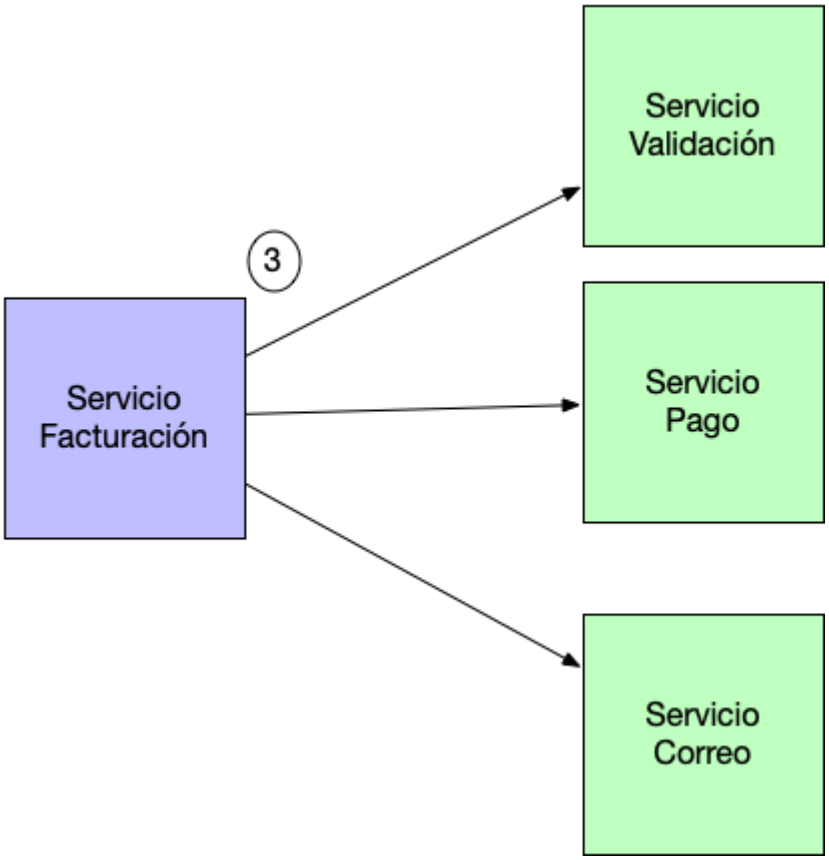
```
      ServicioPago servicioPago) {
    super();
    this.servicioValidacion = servicioValidacion;
    this.servicioCorreo = servicioCorreo;
    this.servicioPago = servicioPago;
  }
  public void procesarFactura(Factura f , String email) {
    servicioValidacion.validarFactura(f);
    servicioPago.pagarFactura(f, "111-111-111");
    servicioCorreo.enviarMensaje(email, "su factura con numero:"+
f.getNumero()+ "ha sido pagada");
  }
}
```

En este caso estamos usando un ServicioFacturación que procesa la Factura , la valida paga y envía un correo confirmando que la Factura ha sido pagada. En principio el código es bastante razonable pero si mostramos un diagrama de clases nos daremos cuenta que estamos ante una relación de este tipo.

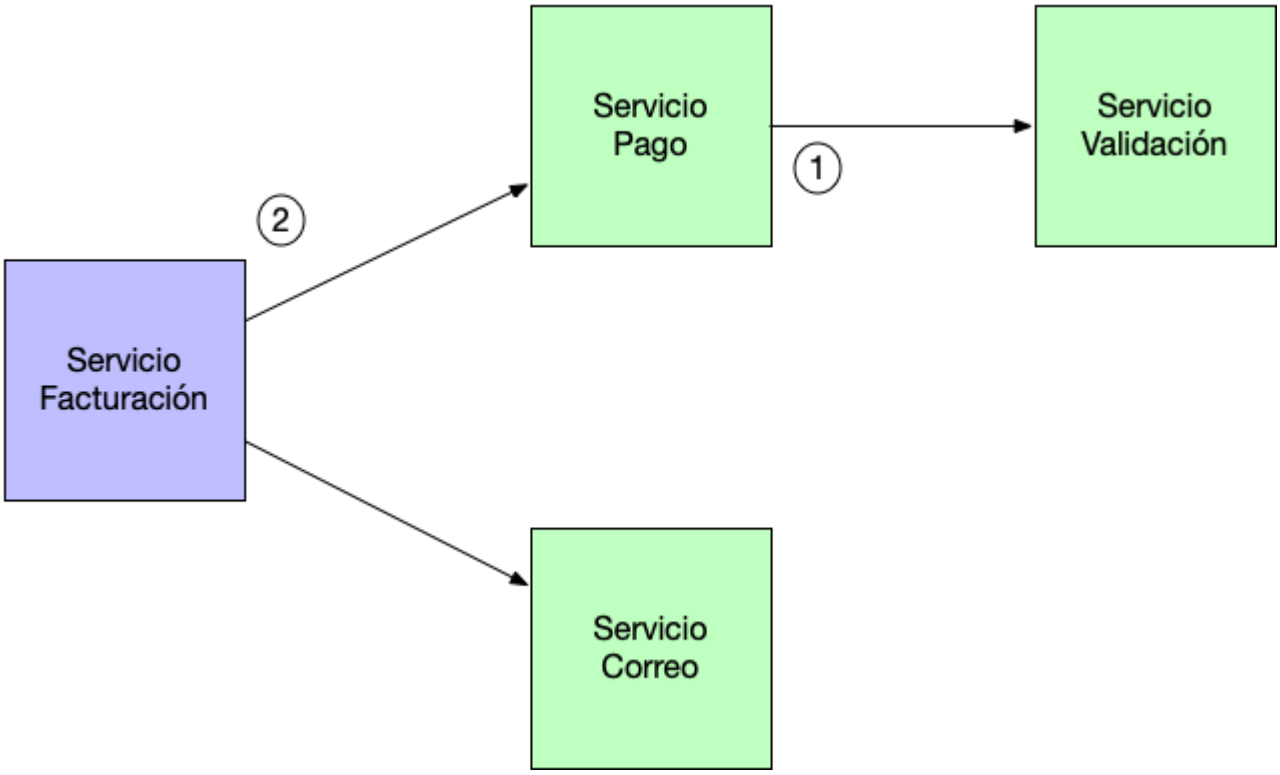


Acoplamiento y clases

El servicioFacturación esta ligado a tres Servicios adicionales , el primero valida , el segundo Paga la factura y el tercero envía un correo con el pago realizado. Esto es lo que se denomina acoplamiento entre clases . Nuestra clase ServicioFacturación depende de otros servicios y por lo tanto cada cambio que realicemos en cualquiera de estas clases implica que el ServicioFacturación también se verá afectado.



¿Podemos mejorar de alguna forma el diseño?. En este caso no es muy complicado ya que probablemente la Factura debe ser validada antes de que realicemos el Pago . Así pues podríamos diseñar la relación de clases de la siguiente forma:



Hemos conseguido reducir el acoplamiento entre clases ya que ahora el servicioFacturación únicamente esta ligado a dos dependencias y su acoplamiento se ha reducido . Esta es una de las características que nuestro código debe cumplir el bajo acoplamiento (low coupling) . Veamos el código:

```
package com.arquitecturajava.serviciosb;

import com.arquitecturajava.Factura;

public class ServicioPago {

  public ServicioValidacion servicioValidacion;

  public ServicioPago(ServicioValidacion servicioValidacion) {
    super();
```

```
    this.servicioValidacion = servicioValidacion;
  }

  public void pagarFactura(Factura f, String cuenta) {
    if (servicioValidacion.validarFactura(f)) {
      System.out.println("pago en cuenta:"+ cuenta);
      System.out.println("importe a pagar"+ f.getImporte());
    }
  }
}
```

```
package com.arquitecturajava.serviciosb;

import com.arquitecturajava.Factura;

public class ServicioFacturacion {


  private ServicioCorreo servicioCorreo;
  private ServicioPago servicioPago;
  public ServicioFacturacion( ServicioCorreo servicioCorreo,
      ServicioPago servicioPago) {
    super();
    this.servicioCorreo = servicioCorreo;
    this.servicioPago=servicioPago;
  }
  public void procesarFactura(Factura f , String email) {
    servicioPago.pagarFactura(f, "111-111-111");
    servicioCorreo.enviarMensaje(email, "su factura con numero:"+
f.getNumero()+ "ha sido pagada");
  }
}
```

Acabamos de refactorizar el código para obtener un bajo acoplamiento

Otros artículos relacionados

- Curso Java Diseño Orientado a Objeto
- DTO y Inmutabilidad
- ¿Framework vs Frameworkless y cuál elegir?
- JPA Polymorphic Query