



**EDUCACIÓN
EN LÍNEA**



Computación Paralela y Distribuida

Inyección de dependencias

Grupo N°2: Carolyn Quilca, Dario Rodríguez

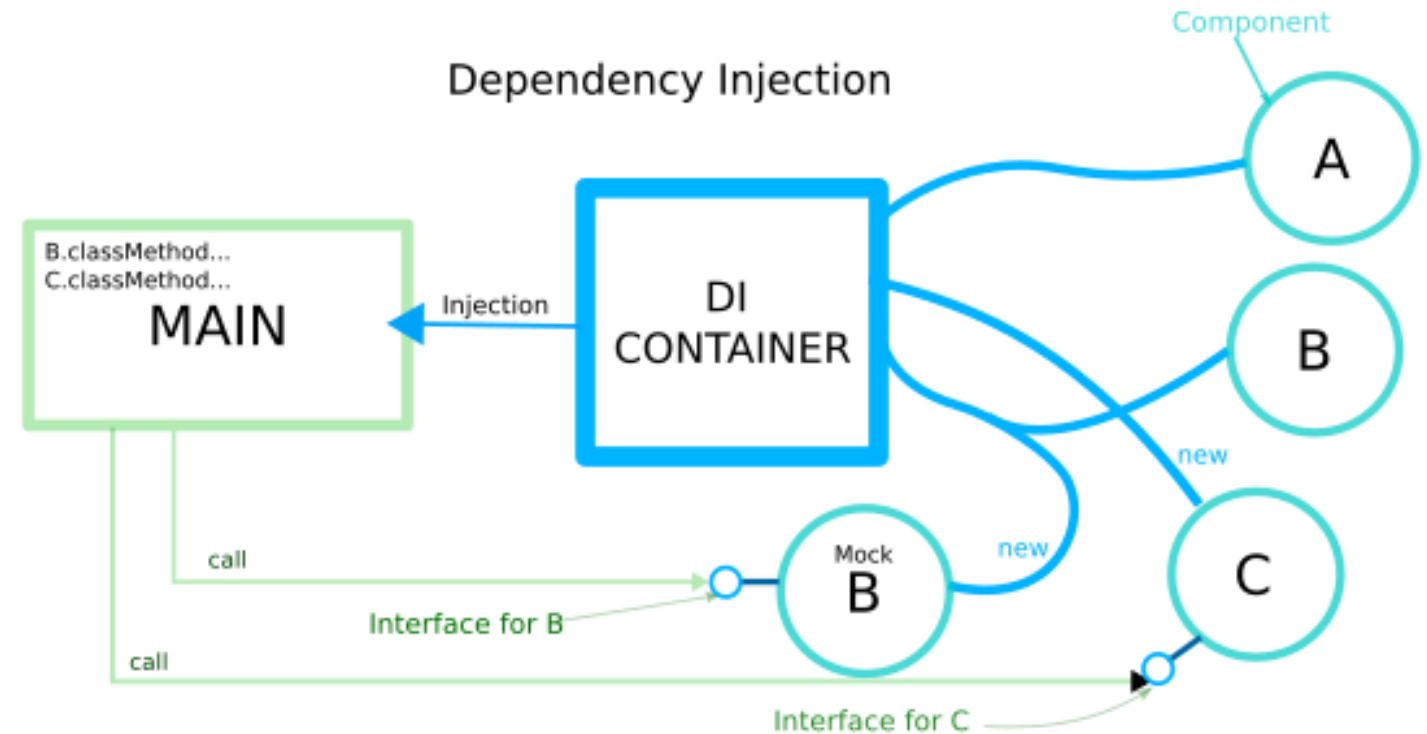
Inyección de dependencias

DI(Dependency Injection)

- Patrón de diseño de software usado para POO.
- Solucionar necesidades de creación de los objetos de una manera práctica, útil, escalable y con una alta versatilidad del código.
- Separar nuestro código por responsabilidades(creación de los objetos).
- En POO tenemos una posible separación del código en dos partes; creación los objetos y uso.
- DI nos permite inyectar comportamientos a componentes haciendo que nuestras piezas de software sean independientes y se comuniquen únicamente a través de una interface(Inversión de Control).

Inyección de dependencias

DI(Dependency Injection)



Inyección de dependencias

DI(Dependency Injection)

- En los comienzos de la programación, los programas eran lineales y monolíticos.
- Para estructurar el código: la modularidad y la reutilización de los componentes.
- El flujo se complica, saltando de componente a componente, y aparece un nuevo problema: la dependencia (acoplamiento) entre los componentes.
- La problemática con estas viene dada por el grado de acoplamiento que tiene la dependencia con el componente.

Inyección de dependencias

DI(Dependency Injection)

Principio de Inyección de Dependencias:

- A. Las clases de alto nivel no deberían depender de las clases de bajo nivel. Ambas deberían depender de las abstracciones.
- B. Las abstracciones no deberían depender de los detalles. Los detalles deberían depender de las abstracciones.

Inyección de dependencias

DI(Dependency Injection)

Contenedor

- El contenedor inyecta a cada objeto los objetos necesarios según las relaciones de dependencia registradas en la configuración previa.
- Implementado por un framework externo a la aplicación (como Spring entre otros), por lo cual en la aplicación también se utilizará inversión de control al ser el contenedor.
- Nos permite que la instanciación de un objeto, por muchas dependencias que tenga, vuelva a ser tan simple como una llamada a un método.

Inyección de dependencias

DI(Dependency Injection)

Declaración de la inyección

- La inyección de dependencias puede realizarse referenciando a las clases de dichas dependencias.
- Se usan interfaces.
- Abstraer la relación entre una clase A que depende de una clase B sin importar la implementación de cada uno de los dos.

Inyección de dependencias

DI(Dependency Injection)

Implementación

- Por constructor
- Por setter
- Por interfaz

Inyección de dependencias

DI(Dependency Injection)

Constructor Dependency Injection

```
public class ConstructorInjection {  
    private Dependency dep;  
  
    public ConstructorInjection(Dependency dep) {  
        this.dep = dep;  
    }  
}
```

Inyección de dependencias

```
*/  
public class SimpleBean {  
  
    private Integer id;  
    private String name;  
    private HashMap<String, Object> maps;  
  
    public void setId(Integer id) {  
        this.id = id;  
    }  
    public void setName(String name) {  
        this.name = name;  
    }  
    public void setMaps(HashMap<String, Object> maps) {  
        this.maps = maps;  
    }  
  
    @Override  
    public String toString() {  
        return "SimpleBean [id=" + id + ", maps=" + maps + ", name=" + name  
            + " ]";  
    }  
}
```

Inyección de dependencias

```
public class PaymentController : Controller
{
    private readonly IPaymentServices _paymentServices;
    private readonly INotificationServices _notificationServices;

    public PaymentController(IPaymentServices paymentServices,
                             INotificationServices notificationServices)
    {
        _paymentServices = paymentServices;
        _notificationServices = notificationServices;
    }

    public IActionResult Cancel(string paymentId)
    {
        _paymentServices.CancelPayment(paymentId);
        _notificationServices.NotifyPaymentCancellation(paymentId);
        return View();
    }
}
```

Inyección de dependencias

DI(Dependency Injection)

Principio de Hollywood

- Invierte el flujo de control de un sistema en comparación con la programación estructurada y modular.
- Registra una implementación específica para cada tipo de interfaz y retorna una instancia de objeto

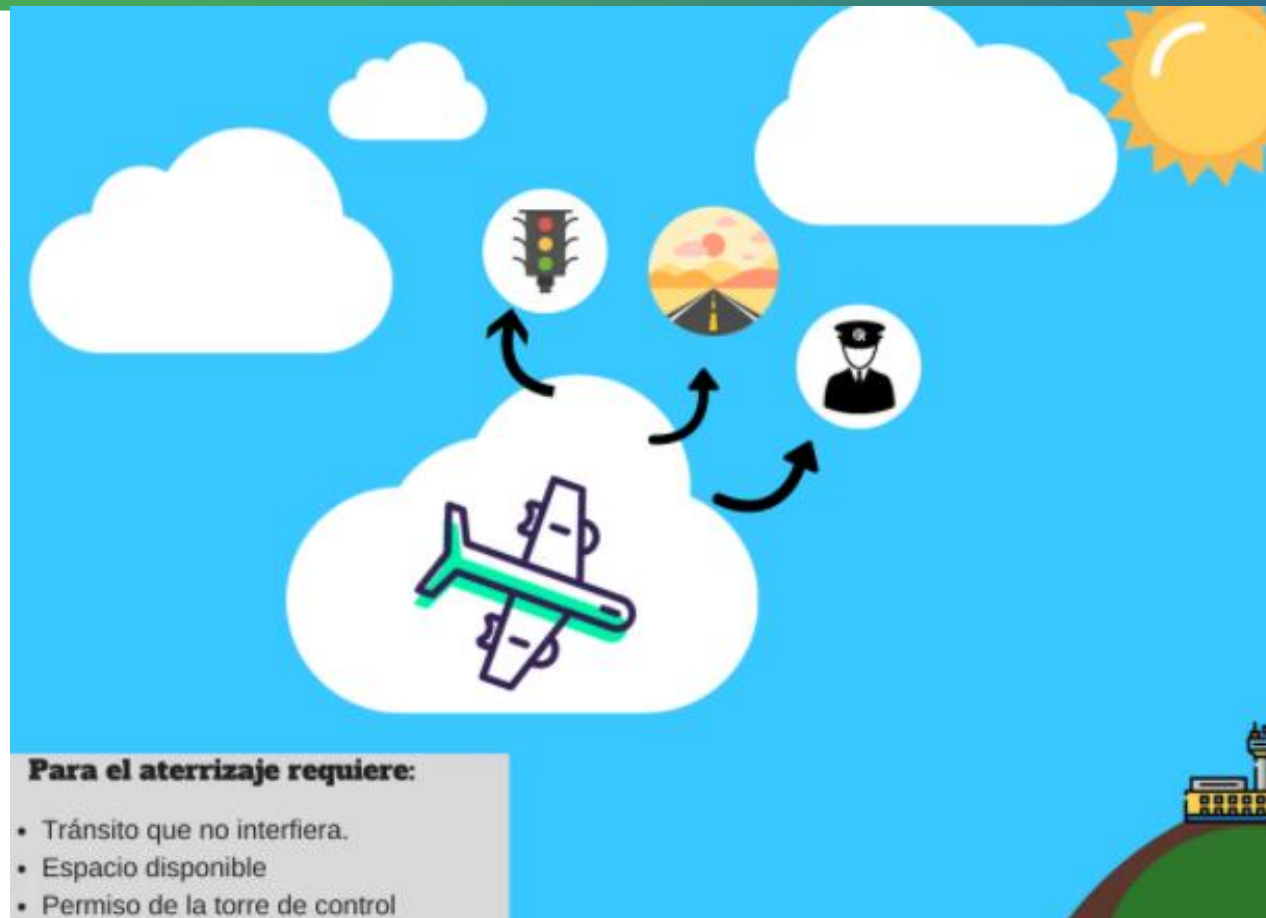
Inyección de dependencias

DI(Dependency Injection)

Ejemplo

- Imagínese que usted como piloto de avión comercial, cada vez que tenga pilotear tuviera que llenar el tanque de combustible, preparar la pista, verificar que todo esta listo para el despegue, sabemos que no es humanamente posible, por eso existen distintas personas trabajan en conjunto para que sea posible el despegue.

Inyección de dependencias



¿Por qué es útil?

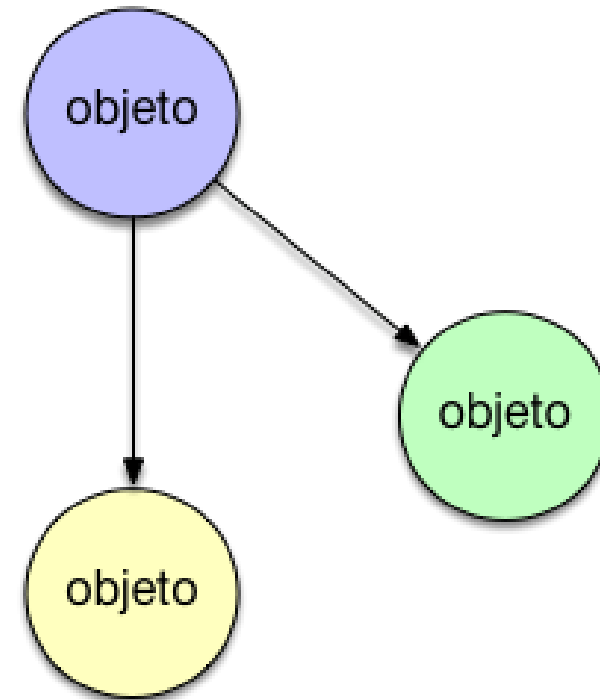
- La DI nos permite cambiar fácilmente el comportamiento de la aplicación al cambiar los componentes que implementaste en las interfaces.
(Elimina acoplamiento entre componentes)
- También resulta que en componentes sea mas fácil de asilar para la prueba unitaria.

Ventajas de usar la DI

- **Flexible** :No hay necesidad de tener un código de búsqueda en la lógica de negocio
- **Testeable** : Testeo automático como parte de las construcciones
- **Mantenible**: Permite la reutilización en diferentes entornos de aplicaciones modificando los archivos de configuración en lugar del código.

¿Para qué sirve este patrón de diseño y cual es su utilizad?

- Normalmente cuando nosotros programamos en el día a día con la programación orientada a objetos nos encontramos construyendo objetos y relacionando objetos utilizando dependencias.



Ejemplo

Por ejemplo podemos tener un programa principal que use un sencillo servicio de impresión para imprimir un documento.

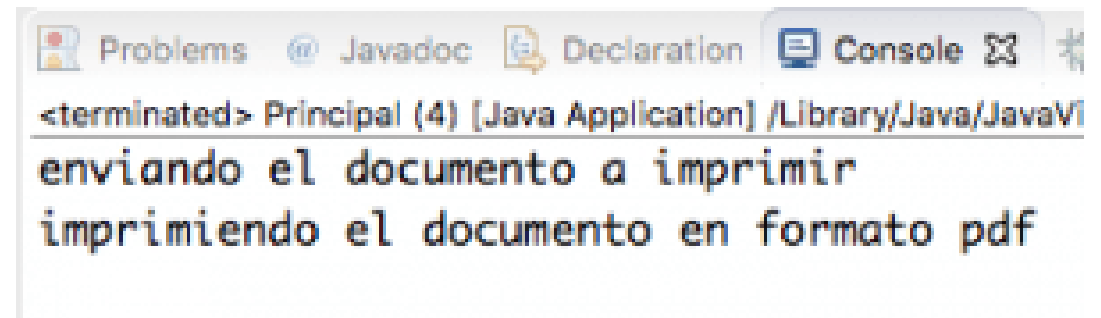
```
public class ServicioImpresion {  
  
    public void imprimir() {  
        System.out.println("enviando el documento a imprimir");  
        System.out.println("imprimiendo el documento en formato pdf");  
    }  
}
```

Ejemplo

Utilizamos un programa main e imprimimos:

```
public class Principal {  
    public static void main(String[] args) {  
        ServicioImpresion miServicio= new ServicioImpresion();  
        miServicio.imprimir();  
    }  
}
```

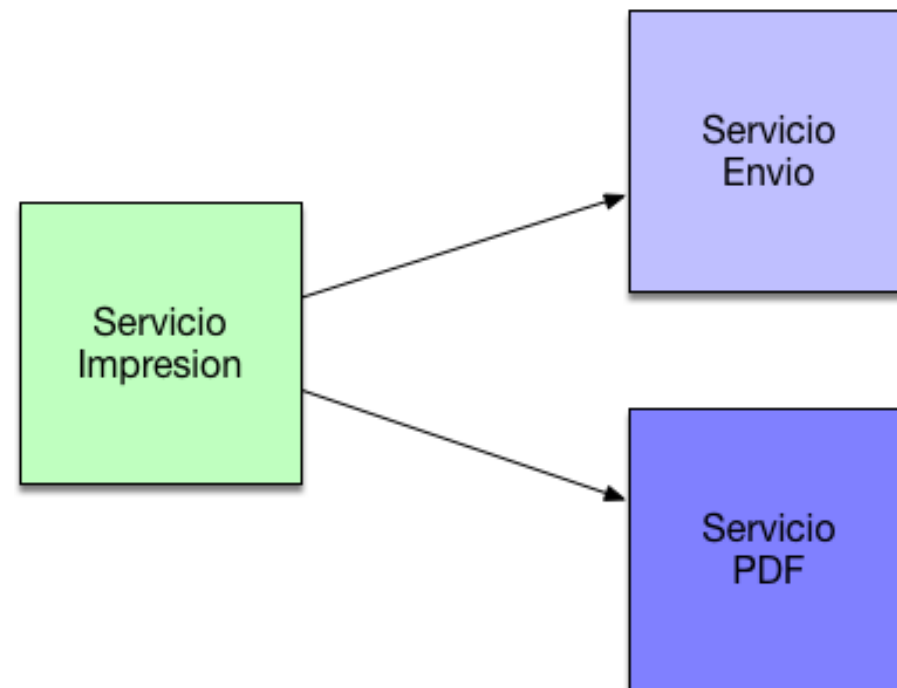
Hasta ahí no tiene nada de especial y veremos impreso el resultado en la consola:



The screenshot shows an IDE console window with tabs for Problems, Javadoc, Declaration, and Console. The Console tab is active, displaying the output of the Java application. The text in the console is: `<terminated> Principal (4) [Java Application] /Library/Java/JavaV` followed by two lines of text: `enviando el documento a imprimir` and `imprimiendo el documento en formato pdf`.

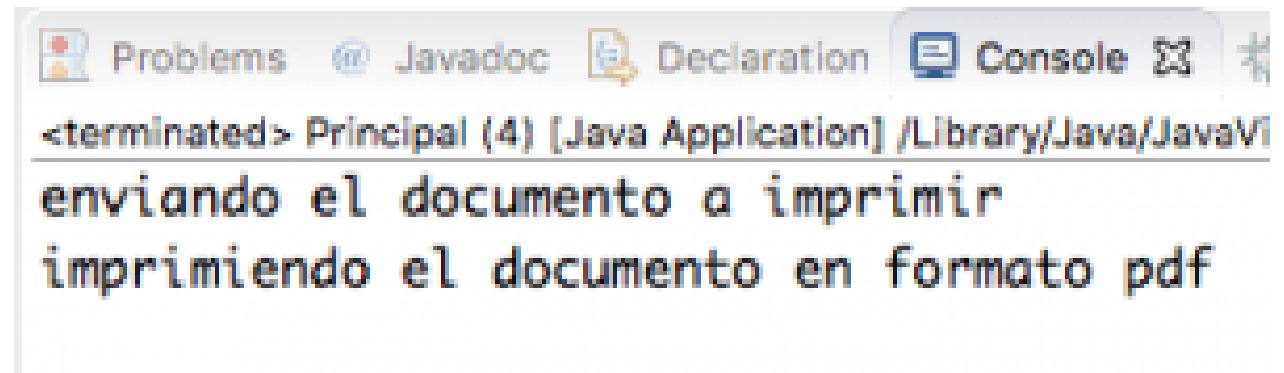
Ejemplo Inyección de dependencia

Lo lógico es que este programa divida un poco más sus responsabilidades y este compuesto de varios servicios algo como lo siguiente:



Ejemplo Inyección de dependencia

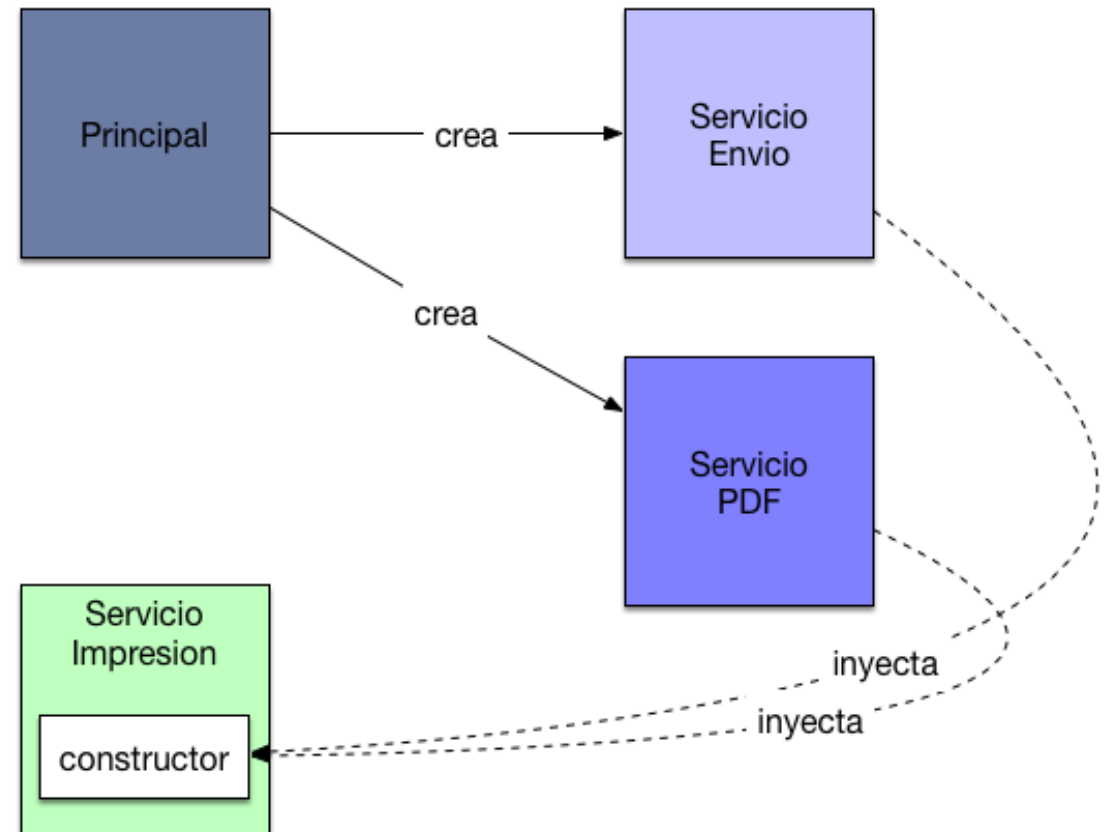
Conseguimos es que nuestro servicio de impresión **dependa de otros servicios y las responsabilidades queden mas claras.**



```
<terminated> Principal (4) [Java Application] /Library/Java/JavaV  
enviando el documento a imprimir  
imprimiendo el documento en formato pdf
```

Ejemplo Inyección de dependencia

Se puede realizar la misma operación inyectando las dependencias al ServicioImpresión y que no sea él el que tenga que definirlas en el constructor.



Ejemplo Inyección de dependencia

Que ventajas aporta esto?

Ya no es el propio servicio el responsable de definir sus dependencias sino que lo es el programa principal.

Esto abre las puertas a la extensibilidad.

```
public static void main(String[] args) {  
  
    //ServicioImpresion miServicio= new ServicioImpresion();  
    ServicioImpresion miServicio = new ServicioImpresion(new ServicioEnvio(), new ServicioPDF());  
    miServicio.imprimir();  
}
```


Ejemplo Inyección de dependencia

Que ventajas aporta esto?

Podemos cambiar el tipo de dependencia que inyectamos ,
simplemente extendiendo **una de nuestras clases y**
cambiando el comportamiento

Ejemplo Inyección de dependencia

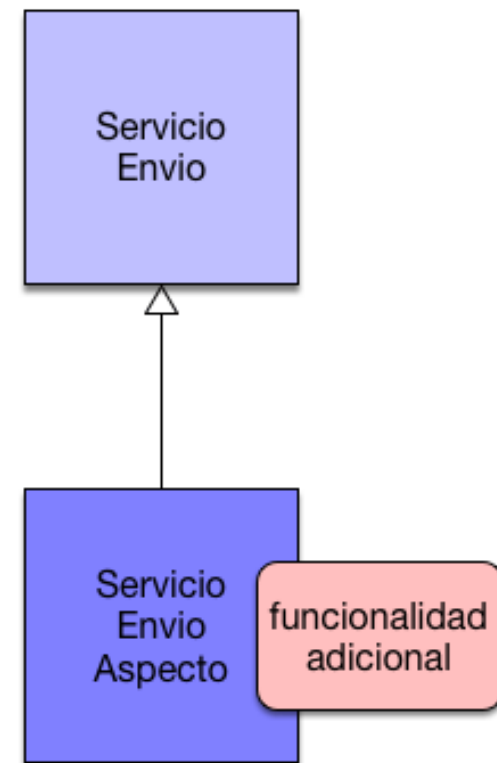
```
public class ServicioEnvioAspecto extends ServicioEnvio {  
    @Override  
    public void enviar() {  
        System.out.println("haciendo log del correo que vamos a enviar");  
        super.enviar();  
    }  
}
```

Acabamos de crear una clase que extiende ServicioEnvio y añade una funcionalidad adicional de log que hace un “log” del correo que enviamos.

Ejemplo Inyección de dependencia

De esta forma habremos cambiado el comportamiento de forma considerable.

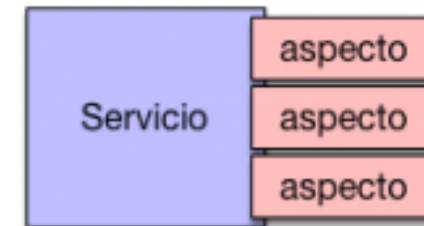
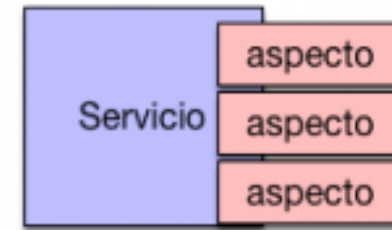
Acabamos de modificar el comportamiento de nuestro programa de forma significativa gracias al uso del concepto de inyección de dependencia.



Conclusión

La inyección de dependencia **nos permite inyectar otras clases y añadir funcionalidad transversal a medida.**

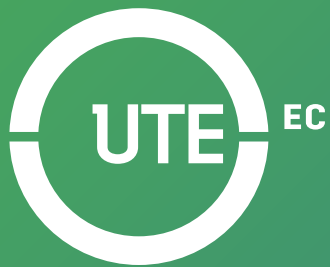
Este patrón de diseño es el que abre la puerta a frameworks como Spring utilizando el concepto de inyección de dependencia de una forma más avanzada.



Inyección de dependencias

Fuentes:

- <http://www.tss.com.pe/blog/que-es-la-inyeccion-de-dependencias-para-que-sirve-y-que-significan-los-tiempos-de-vida-en-su-implementación>
- [https://es.wikipedia.org/wiki/Inyecci%C3%B3n de dependencias](https://es.wikipedia.org/wiki/Inyecci%C3%B3n_de_dependencias)
- <https://desarrolloweb.com/articulos/patron-diseno-contenedor-dependencias.html>
- <http://dominiotic.com/que-es-la-inyeccion-de-dependencias/>
- <https://www.arquitecturajava.com/el-patron-de-inyeccion-de-dependencia/>
- <https://dev.to/cchacin/inyeccion-de-dependencias-en-java-44cg>



¡GRACIAS!

**TRAS
CENDE
MOS**

A white curved line graphic, resembling a stylized 'C' or a partial arc, positioned to the right of the text.