

**FACULTAD DE CIENCIAS E INGIENERIA**

**ESCUELA ACADÉMICA DE INGIENERIA DE SISTEMAS**

**TEMA**

**PATRON INYECCIÓN DE DEPENDENCIAS**

**CURSO DE:**

**PROGRAMACION I**

**Propósito del trabajo: Primera Parte**

CORONEL CASTILLO ERIC GUSTAVO

AUTOR(A):

Jara Monge Luis Antonio

Roca Ulloa Richard

Vázquez Torres Gilson Anderson

Los Olivos – Perú

2017 – II

“Año del buen servicio al Ciudadano”

Este trabajo está elaborado y dedicado a todo el apoyo de nuestros padres que se esfuerzan para lograr nuestros objetivos y seguir con nuestras metas.

**Índice**

[**Índice** 3](#_Toc495010613)

[**Resumen** 4](#_Toc495010614)

[**Antecedentes** 5](#_Toc495010615)

[**DESARROLLO DEL TEMA** 6](#_Toc495010616)

[**CAPÍTULO I. Inyección de Dependencias** 6](#_Toc495010617)

[**1.1.** **¿Qué es la Inyección de Dependencias?** 6](#_Toc495010618)

[**1.2.** **Ventajas** 6](#_Toc495010619)

[**CAPÍTULO II: Tipos de Implementación** 7](#_Toc495010620)

[**1.** **Inyección basada en métodos setters.** 7](#_Toc495010621)

[**2.** **Inyección basada en constructor** 9](#_Toc495010622)

[**3.** **Inyección basada en interfaces** 10](#_Toc495010623)

[**Conclusiones** 11](#_Toc495010624)

[**Recomendaciones** 11](#_Toc495010625)

[**Referencias Bibliográficas** 12](#_Toc495010626)

**Resumen**

La Inyección de dependencias es un patrón de más genérico llamado inversión de control. DI hace uso de la modularidad y reutilización, para entender mejor la DI más adelante veremos un ejemplo de esto.

DI, corresponde a las siglas “Dependency Injection”, es un patrón de diseño pensado en permitir un menor acoplamiento entre componentes de una aplicación y fomentar así la reutilización de los mismos

En el desarrollo de software moderno se ha comenzado a generalizar el uso de patrones para implementar partes de nuestros desarrollos. Los patrones proponen soluciones estándares a los problemas comunes de diseño de nuestras aplicaciones.

Lo que propone entonces la inyección de dependencias, es no instanciar las dependencias explícitamente en su clase, sino que declarativamente expresarlas en la definición de la clase. La esencia de la inyección de las dependencias es contar con un componente capaz de obtener instancias validas de las dependencias del objeto y pasárselas durante la creación o inicialización del objeto.

**DESARROLLO DEL TEMA**

**CAPÍTULO I. Inyección de Dependencias**

* 1. **¿Qué es la Inyección de Dependencias?**

Desde hace unos años, con la aparición del framework “**Spring”**, se habla mucho del “patrón **Inyección del Dependencias”**. Al igual que cualquier técnica o herramienta tiene sus ventajas e inconvenientes ya que un uso indebido, pensando que nos va a solucionar nuestros problemas, se puede volver en nuestra contra creando una arquitectura compleja e innecesaria.

El patrón de inyección de dependencias consiste en hacer que nuestras piezas de software sean independientes comunicándose únicamente a través de un interface. Esto implica muchas modificaciones en el código fuente como el uso de implementaciones, la eliminación de la instanciación de objetos mediante la instrucción “**new”** o la necesidad de un modo de configuración que indique que clases se instanciarán en el caso de solicitarlo.(Rubira, 2012)

* 1. **Ventajas**

Las ventajas que nos va a aportar esto es todo lo contrario a los problemas que teníamos con dependencias directas, es decir, nuestros objetos no están acoplados, si tengo una objeto B que depende de una abstracción de A, voy a poder sustituir la implementación de A sin tocar el objeto B.

Vamos a poder trabajar de forma más independiente entre compañeros, porque si yo me encargo de realizar el objeto B, no necesito que la implementación de A este codificado para poder trabajar. Nos facilitara al realizar las pruebas unitarias.

**CAPÍTULO II: Tipos de Implementación**

Maneras de implementar la inyección de dependencias:

1. **Inyección basada en métodos setters.**

En este tipo de DI, se utiliza el método setters para inyectar una dependencia en el objeto que requiere. Se invoca así el setter de cada dependencia y de le pasa como parámetro una referencia a la misma

Considerando un diseño en dos capas donde tenemos una capa service donde se implementa los componentes de negocio, y una capa POR donde se implementa los componentes de persistencia, se tiene una clase VentaService y una interface VentaPorApli, VentaService depende de VentaPorApli para operar correctamente.

VentaPorApli

VentaService

VentaPorPython

VentaPorHtml

Venta service necesita que le inyecten una implementación de la interface VentaPorApli, a continuación se mostrara el código:

1. Interface VentaPorApli{
2. }

Código de interface VentaPorApli.

1. Class VentaPorHtml implements VentaPorPython{
2. }
3. Class VentaPorPython implements VentaPorApli{
4. }
5. **Inyección basada en constructor.**
6. **Inyección basada en interfaces.**

Implementación para VentaPorHtml y VentaPorPython.  
En VentaService tenemos la variable de tipo ventaPorApli, la variable es ventaPor, a través del método setter se le inyecta la dependencia.

1. Public class VentaService{
2. Private VentaPorApli ventaPor;
3. Public void setVentaPor(VentaPorApli ventaPor){
4. This.ventaPor = ventaPor;
5. }
6. }

El objeto responsable de las dependencias realizara la inyección de la siguiente forma

1. VentaPorApli ventaPor = new VentaPorPython();
2. VentaService VentaService = new VentaService();
3. VentaService. VentaPor(ventaPor);

Se instancia una implementación de ventaPorApli luego se instancia VentaService y luego se inyecta ventaPor, una vez que el objeto venta service esta creado e inicializado el componente builder da la referencia a ese objeto.

1. **Inyección basada en constructor**

Las dependencias se inyectan utilizando un constructor con parámetros del objeto dependiente. Este constructor recibe las dependencias como parámetros y las establece en los atributos del objeto.  
En la inyección basada en constructor se creara una instancia de VentaService usando un constructor con parámetros el cual se le pasara una referencia a un objeto de clase que implemente VentaPorApli para poder inyectar la dependencia.

1. public class VentaService{
2. private VentaPorApli ventaPor;
3. public VentaService(VentaPorApli ventaPor){
4. this.ventaPor = ventaPor;
5. }
6. }

Constructor con parámetros, se crea un constructor de venta service que tenga como paramento una variable de tipo VentaPorApli, y al ejecutar se pasa la implementación que se corresponde sea por Html o Python.

1. VentaPorApli = ventaPor = new VentaPorPython();
2. VentaService ventaService = new VentaService(VentaPor);

Se instancia ventaPorApli se implementa para VentaPorPython y cuando se crea la instancia a través de venta service a través del parámetro del constructor se le pasa el objeto ventaPor de tal manera que VentaService queda inicializado para ser reutilizado.

1. **Inyección basada en interfaces**

En este caso se utiliza una interface común que otras clases la implementen para poderles luego inyectar dependencias. Toda clase que implemente VentaServiceApli se le podrá inyectar cualquier objeto que implemente VentaPorApli mediante el método injectVentaPor.

1. interface VentaServiceApli{
2. void injectVentaPor(VentaPorApli ventaPor);
3. }
4. class VentaService implements VentaServiceApli{
5. private VentaPorApli ventaPor;
6. public void injectVentaPor (VentaPorApli ventaPor){
7. this.ventaPor = ventaPor;
8. }
9. }

Cualquier clase que quiera inyectarse se tendrá que implementar esta interface, y a través de este método se implementara la dependencia, se tendrá que crear una interface y además tendrá que implementar la clase donde se va a ser la inyección.

1. VentaPorApli ventaPor = new VentaPorPython();
2. VentaService ventaService = new VentaService();
3. ventaService.inyectVentaPor(ventaPor);

Cuando se crea esa instancia el componente builder crea primero la instancia de ventaPorApli luego la instancia de VentaService y luego a través del método que define la interface se le inyecta su dependencia.

**Conclusiones**

Las formas de inyectar dependencias nos ayudaran a tener un menor acoplamiento en nuestro código. Cuando el acoplamiento se realiza a una interface, se puede utilizar cualquier implementación con el mínimo cambio.

**Recomendaciones**

La inyección de dependencias no es necesario si se va a utilizar la misma implementación de una dependencia o la misma configuración.

Se recomienda aplicar inyecciones de dependencias para:

* Inyectar diferentes implementaciones de la misma dependencia.
* Inyectar la misma implementación en varias configuraciones.
* Inyectar datos de configuración de un componente.
* Inyectar la misma dependencia en varios componentes.

**Referencias Bibliográficas**

<http://xurxodeveloper.blogspot.pe/2014/09/Inyeccion-de-Dependencias-DI.html>

<https://es.wikipedia.org/wiki/Inyecci%C3%B3n_de_dependencias>

<https://www.genbetadev.com/paradigmas-de-programacion/que-es-la-inyeccion-de-dependencias>

<https://www.adictosaltrabajo.com/tutoriales/dependency-injector/>

<https://www.youtube.com/watch?v=XBQ_asYbnZw&t=1229s>