

**FACULTAD DE INGENIERIA**

**INTEGRANTES:**

Paico Campos Meyluz

Hidalgo Junior

**TEMA:**

Patrón Inyección de Dependencias

**CICLO:**

III

**PROFESOR:**

Eric Gustavo Coronel

**INDICE**

* 1. **INTRODUCCION…………………………………………………………………………….3**
  2. **¿QUE ES?.........................................................................................3**
  3. **Diseñando nuestro patrón de inyección de dependencias……………..4**
     1. **INYECTOR DE DEPENDENCIAS……………………………………………………4**
     2. **CONFIGURACION DE DEPENDECIAS…………………………………………..4**
     3. **CREADOR DE INSTANCIAS………………………………………………………….4**
     4. **CONFIGURACION DE INSTANCIAS………………………………………………4**
  4. **Esquema de funcionamiento………………………………………………………….5**
  5. **Implementación del patrón en Java………………………………………………..5**
  6. **El Principio de Inyección de Dependencias……………………………………..6**

**1.7.1 Martin Fowler………………………………………………………………………………7**

**1.7.2 Biografía………………………………………………………………………………………7**

**1.8Ejemplos…………………………………………………………………………………....……8**

**1.9 Conclusión……………………………………………………………………………………..10**

**1.10 Referencias……………………………………………………………………………….…11**

**1.1 INTRODUCCION:**

La inyección de dependencias es un patrón de diseño de software usado en la Programación Orientada a Objetos, que trata de solucionar las necesidades de creación de los objetos de una manera práctica, útil, escalable y con una alta versatilidad del código.

En la mayoría de los frameworks actuales se aplica la Inyección de dependencias como parte de las herramientas y modelos que facilitan al programador. Como cualquier patrón de diseño de software trata de solucionar de una manera elegante un problema habitual en el desarrollo de software, por lo que también es idóneo utilizar este patrón en el desarrollo de proyectos a pequeña escala.

**1.2 ¿QUE ES?**

Aparte de un patrón de diseño de software, nos ayuda a separar nuestro código por responsabilidades, siendo que en esta ocasión sólo se dedica a organizar el código que tiene que ver con la creación de los objetos.

Sabemos, uno de los uno de los principios básicos de la programación, y de las buenas prácticas, es la separación del código por responsabilidades. Pues la inyección de dependencias parte de ahí.

En el código de una aplicación con OOP (Programación Orientada a Objetos) tenemos una posible separación del código en dos partes, una en la que creamos los objetos y otra en la que los usamos. Existen patrones como las factorías que tratan esa parte, pero la inyección de dependencias va un poco más allá. Lo que dice es que los objetos nunca deben construir aquellos otros objetos que necesitan para funcionar. Esa parte de creación de los objetos se debe hacer en otro lugar diferente a la inicialización de un objeto.

**1.3 Diseñando nuestro patrón de inyección de dependencias**

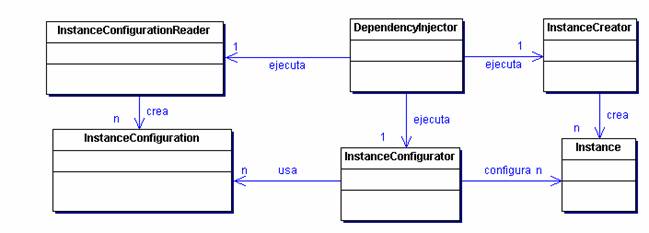
Podemos crear una versión sencilla de este patrón utilizando un diseño bastante simple.

Este patrón lo vamos a descomponer en los siguientes objetos:

**1.3.1 INYECTOR DE DEPENDENCIAS**: Es el objeto que controla cómo se realiza la inyección de dependencias. Utiliza los objetos lector de configuración de dependencias, creador de instancias e configuración de instancias. La configuración de las instancias la almacena en el mapa de configuración de instancias y las instancias en el mapa instancias.

**1.3.2. CONFIGURACION DE DEPENDECIAS:** Este objeto lee la configuración de dependencias que se va a utilizar. Esta configuración está compuesta de una serie de objetos que se deben instanciar, la lista de parámetros con sus valores que se debe proporcionar a cada objeto para su configuración y las relaciones que se deben crear entre los diferentes objetos.

**1.3.3. CREADOR DE INSTANCIAS:** Este es un objeto que sigue el patrón factoría, y se ocupa de crear instancias y ponerlas a disposición de los demás objetos del patrón.

**1.3.4. CONFIGURACION DE INSTANCIAS:** Este objeto conoce cómo se debe configurar cada uno de los objetos instanciados, asignándole los parámetros y relaciones.

En el diagrama se puede ver el diseño básico del patrón. La clase principal Inyector de dependencia sirve como interfaz para el conjunto. El usuario interactuará con el conjunto a través de esta clase. Para diseñar esta clase podemos usar el patrón singleton, ya que normalmente nos hará falta una única instancia de la misma.

**1.4 Esquema de funcionamiento**

Al instanciar un inyector de dependencias, este objeto usa el objeto lector de configuración de dependencias para que éste obtenga la información de dependencias a utilizar. A continuación se instancia un objeto creador de instancias para que se puedan crear las instancias de objetos solicitadas. Luego el Inyector de dependencias crea una instancia de la configuración de dependencias y le pasa a esta instancia las referencias a los otros objetos instanciados (inversión de control en esta instancia, como se puede ver). Ahora es el configuración de dependencias el responsable de ir creando instancias de objetos, mediante el InstanceCreator, y luego irlas configurando, según las instrucciones que reciba del configuración de dependencias.

**1.5 Implementación del patrón en Java**

La forma habitual de implementar este patrón es mediante un "Contenedor DI" y objetos planos o simples por ejemplo los llamados POJO en Java. El contenedor inyecta a cada objeto los objetos necesarios según las relaciones plasmadas en un fichero de configuración.

Típicamente este contenedor es 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 (almacenado en una biblioteca) quien invoque el código de la aplicación. Ésta es la razón por la que los términos de inversión de control e inyección de dependencias se confunden habitualmente entre sí.

**1.6 El Principio de Inyección de Dependencias**

Robert C. Martin afirma en el 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.*

Existen tres formas de implementación de la Inyección de Dependencias:

* **Por constructor:**
* **Por setter:**
* **Por interfaz:** El acoplamiento será menor

**1.7.1 Martin Fowler**

Es un [ingeniero de software](https://es.wikipedia.org/wiki/Ingenier%C3%ADa_de_software) [británico](https://es.wikipedia.org/wiki/Gran_Breta%C3%B1a), autor y orador internacional sobre desarrollo de software, especializado en análisis y diseño [orientado a objetos](https://es.wikipedia.org/wiki/Programaci%C3%B3n_orientada_a_objetos), [UML](https://es.wikipedia.org/wiki/Lenguaje_Unificado_de_Modelado), [patrones de diseño](https://es.wikipedia.org/wiki/Patr%C3%B3n_de_dise%C3%B1o), y metodologías de [desarrollo ágil](https://es.wikipedia.org/wiki/Desarrollo_%C3%A1gil_de_software), incluyendo [programación extrema](https://es.wikipedia.org/wiki/Programaci%C3%B3n_extrema).

**1.7.2 BIOGRAFIA**

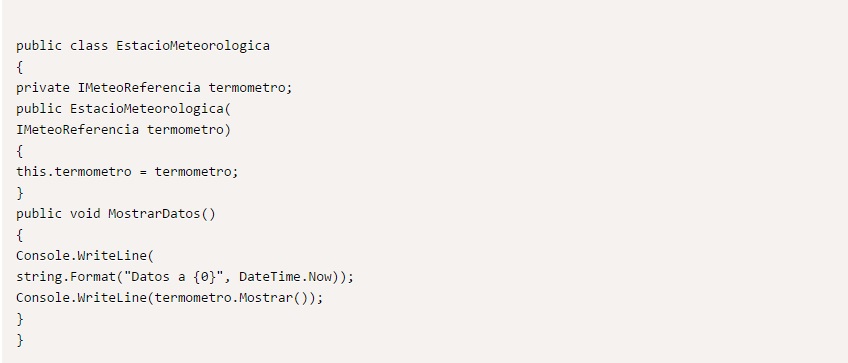
Fowler nació y creció en [Walsall](https://es.wikipedia.org/wiki/Walsall), [Inglaterra](https://es.wikipedia.org/wiki/Inglaterra), donde asistió a [Queen Mary's Grammar School](https://es.wikipedia.org/w/index.php?title=Queen_Mary%27s_Grammar_School&action=edit&redlink=1) como escuela secundaria. Se graduó en [University College de Londres](https://es.wikipedia.org/wiki/University_College_de_Londres) en 1986. En 1994 se mudó a [Estados Unidos](https://es.wikipedia.org/wiki/Estados_Unidos), donde vive cerca de [Boston](https://es.wikipedia.org/wiki/Boston), [Massachusetts](https://es.wikipedia.org/wiki/Massachusetts) en el suburbio de [Melrose](https://es.wikipedia.org/wiki/Melrose_(Massachusetts)).

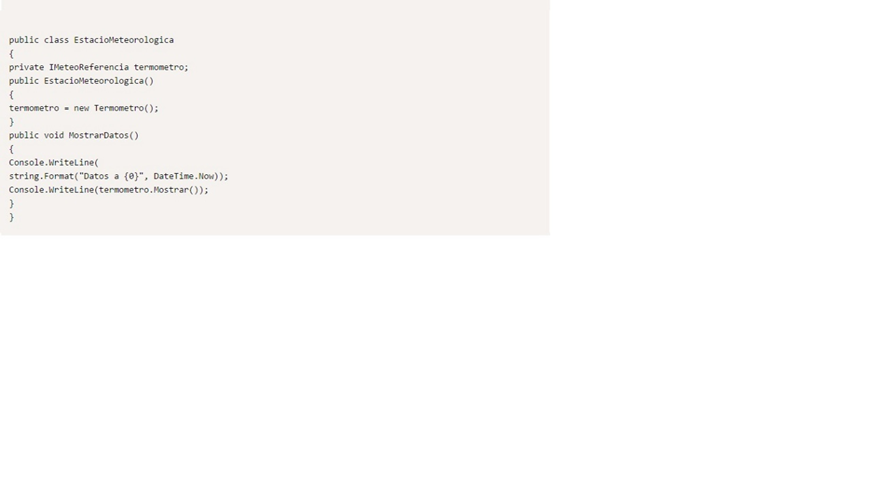
Comenzó a trabajar con software en los inicios de los años 1980. En 1986, una vez terminada la secundaria comenzó a trabajar en desarrollo de software en [Coopers & Lybrand](https://es.wikipedia.org/wiki/Coopers_%26_Lybrand) hasta 1991. En el año 2000 se convirtió en Chief Scientist en [ThoughtWorks](https://es.wikipedia.org/w/index.php?title=ThoughtWorks&action=edit&redlink=1), una compañía de integración de sistemas y consultoría.

Fowler ha escrito siete libros sobre el tema Desarrollo de software (ver [*Publicaciones*](https://es.wikipedia.org/wiki/Martin_Fowler#Publicaciones)). Es miembro de [*Agile Alliance*](https://es.wikipedia.org/w/index.php?title=Agile_Alliance&action=edit&redlink=1) y ayudó a crear el [Manifiesto para el desarrollo ágil de software](https://es.wikipedia.org/w/index.php?title=Manifiesto_para_el_desarrollo_%C3%A1gil_de_software&action=edit&redlink=1) en 2001, junto con más de 15 coautores.

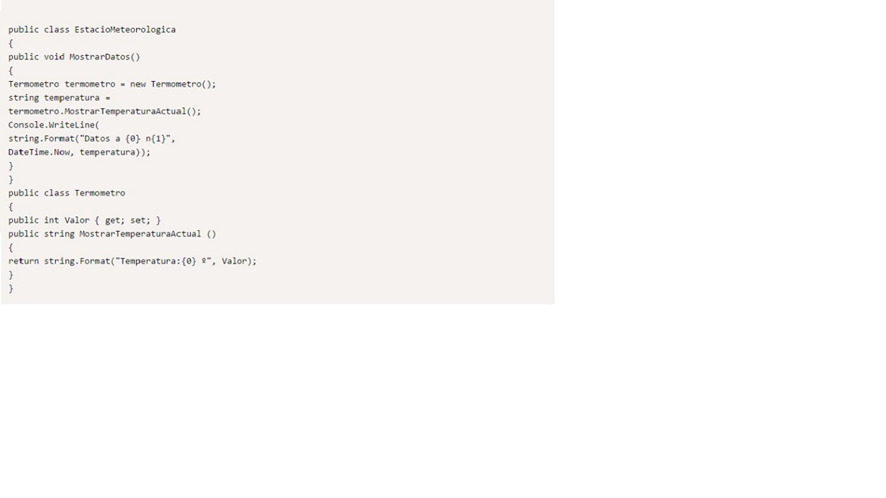
Él mantiene un [*bliki*](https://es.wikipedia.org/w/index.php?title=Bliki&action=edit&redlink=1), una mezcla de [blog](https://es.wikipedia.org/wiki/Blog) y [wiki](https://es.wikipedia.org/wiki/Wiki). Popularizó el término [Inyección de dependencias](https://es.wikipedia.org/wiki/Inyecci%C3%B3n_de_dependencias) como una forma de [Inversión de control](https://es.wikipedia.org/wiki/Inversi%C3%B3n_de_control).

**1.8 EJEMPLO:**



****

****

****

**1.9 CONCLUSION:**

¿Cuándo usar Inyección de dependencias? Casi Siempre

¿Por Qué? Porque nos ayuda a escribir código que es: fácil de mantener, fácil de extender o modificar, código modular, entendible y limpio.

Mostrar de una forma práctica la relación existente entre dependencias, detalles y abstracciones. Con el Principio de Inyección de Dependencias, ponemos fin a la última de las siglas que componen SOLID. Existen libros íntegros que hablan de este principio, y podrá encontrar en Internet una gran cantidad de recursos relacionados.

**1.10 REFERENCIAS**