Profesor: Eric Gustavo Coronel

Año: 2020

Curso: Programación en Java

Integrantes: Flores Berrocal Anthony Aldair

Rojas Asparrin Antony Daniel

INYECCION DE DEPENDENCIAS

Proyecto de investigación



**DEDICATORIA**

Dedicamos este trabajo a:

Dios, por darnos las virtudes necesarias para llevar a cabo este trabajo y guiarnos en nuestras metas a corto, mediano y largo plazo

Nuestro tutor, por su enseñanza y el profesionalismo que siempre a través del respeto mostró en sus clases.

Nuestros Padres, por sus consejos, paciencia y toda la ayuda que me brindaron para seguir adelante en mis estudios

**ÍNDICE:**

[**INTRODUCCIÓN**](#_htb5oq5xfrgu)4

[1.1. Paradigmas de la programación:](#_itbpp4u6gg4z) 4

[1.1.1. Programación Orientada a Objetos (POO)](#_1alnkl763krb) 4

[1.1.1.1. Objetivo](#_uni1ah7ajti8) 4

[1.1.1.2. Pilares](#_dvgjycmaikz) 5

[1.2. SOLID](#_99hu6wtpnhic) 6

[1.3. Patrones de diseño](#_7xetb51uv0c6) 7

[**DESARROLLO DEL TEMA**](#_hxvsk4cj7bcv)8

[2.1. Contexto](#_f21ez3dgazrb) 8

[2.2. Conceptos Relacionados con la Inyección de Dependencias](#_2qchx76l31po) 8

[2.2.1. Inversión de Control (IoC)](#_40ousfetlkm1) 8

[2.2.2. Patrones de Diseño](#_rx53p851yhsw) 9

[2.2.3. Principio de Inversión de Dependencia](#_59b4mo7afr7z) 9

[2.3. Inyección de Dependencias](#_nlo3e5ooi7eh) 10

[**CASOS DESARROLLADOS**](#_x2jv4guvzw9f)11

[3.1. Formas de Inyectar Dependencias](#_hm1q11snkavp) 11

[3.1.1. En el constructor](#_76cnejohllwi) 11

[3.1.2. A través de un setter](#_t6sn43lfl1nk) 12

[3.1.3. A través de un método](#_vtwc1j5jr3go) 12

[3.1.4. A través de un campo](#_tbp9olxgt0nl) 12

[**CONCLUSIONES**](#_15518rfqmprm)13

[**RECOMENDACIONES**](#_l4e8sudvn3np)13

[**Bibliografía**](#_kcc1w9i1qzsf)13

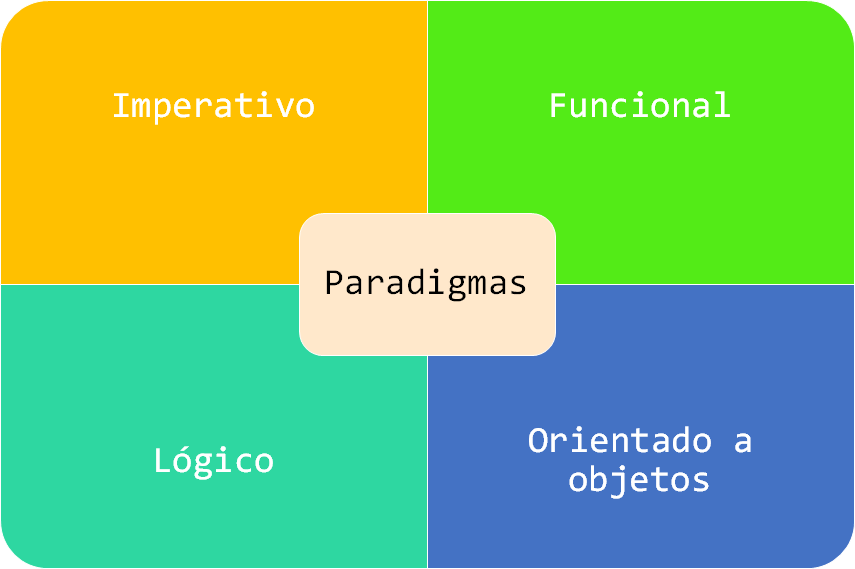
INTRODUCCIÓN

1.1. Paradigmas de la programación:

Para definir que es un paradigma de la programación primero debemos definir que es un paradigma.

Según la RAE, un paradigma es una “teoría o conjunto de teorías cuyo núcleo central se acepta sin cuestionar y que suministra la base y modelo para resolver problemas y avanzar en el conocimiento”, por lo que llevándolo al terreno de la programación, un paradigma de la programación es la aplicación de distintos modelos, metodologías, técnicas y patrones con el fin de llegar a una solución de un problema ligado a la programación.

Existen distintos tipos de paradigmas de la programación entre los cuales están:

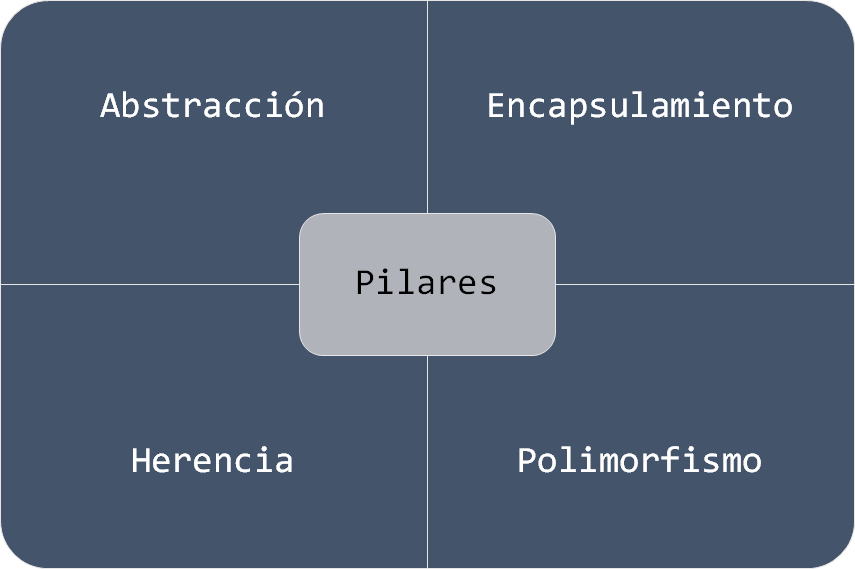
Fuente: Elaboración propia

1.1.1. Programación Orientada a Objetos (POO)

1.1.1.1. Objetivo

Según Platzi (2019), el objetivo de la Programación Orientada a Objetos (POO) es “Orientar a identificar y analizar cada elemento de nuestros sistemas como objetos individuales que tienen atributos y comportamientos”.

1.1.1.2. Pilares

Fuente: Elaboración propia

* Abstracción: Análisis de los objetos para abstraer su composición y generar diferentes clases/moldes/modelos que nos permitan crear todos los objetos de nuestro sistema.
* Encapsulamiento: Definir/Limitar el nivel de acceso que tienen las propiedades y métodos de nuestros objetos, ya que no todas las partes de nuestro programa deben poder leer y/o modificar atributos especiales en nuestras clases.
* Herencia: Crear clases (subclases) a partir de otras clases (superclases) que implementan diferentes comportamientos que podemos reutilizar en diferentes objetos del sistema y evitar código repetido.
* Polimorfismo: Sobrescribir algunos métodos de la clase de la cual heredan nuestras subclases para asignar comportamientos diferentes y funcionar correctamente a pesar de recibir argumentos diferentes (más flexibles) que los que definimos en sus clases padre.

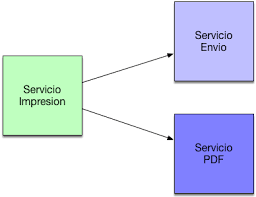
1.2. SOLID

Para poder construir un código más legible, mantenible y escalable debemos guiarnos de los 5 principios SOLID, los cuales son una guía necesaria para las buenas prácticas de programación.

Fuente: *Dominio TIC* (2018)

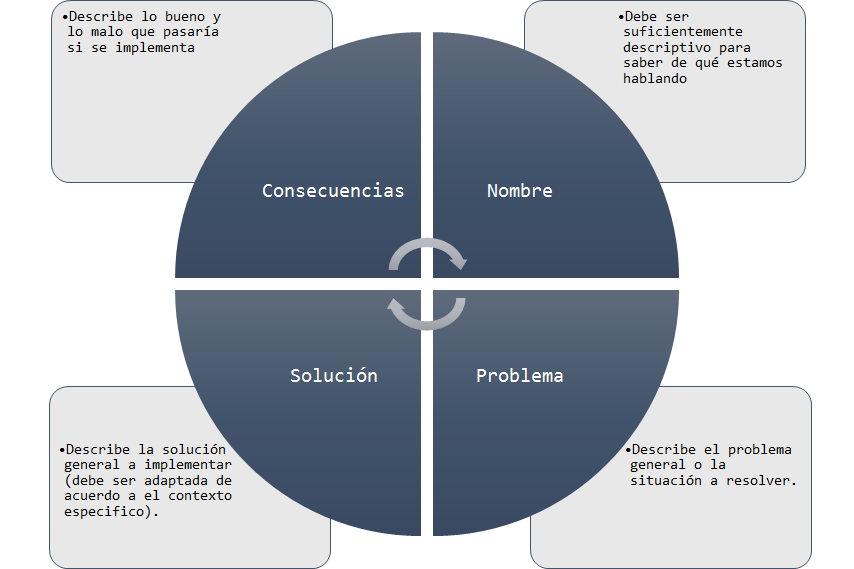
1.3. Patrones de diseño

Para Amaya (2016), un patrón de diseño “es una solución general reusable que puede ser aplicada a problemas que ocurren comúnmente en el desarrollo de software, es la descripción o plantilla de cómo resolver un problema que puede ser usada en diferentes situaciones.”



Fuente: *Arquitectura Java* (2018)

Elementos:



Fuente: *Elaboración propia*

DESARROLLO DEL TEMA

2.1. Contexto

La Inyección de Dependencias es un concepto que hoy en día se ha convertido en uno de los principios más usados en el mundo de desarrollo de aplicaciones. Es igual de importante para el desarrollo de componentes back-end (donde se implementó inicialmente), como de front-end. Angular JS, uno de los frameworks de JavaScript más usados en estos últimos años, y basa su funcionamiento en este principio. Así como también en Spring que tiene mucho éxito hasta la fecha en los programadores de Java. Es por ello que, actualmente, se ha convertido en un requisito indispensable para todo programador entender este patrón y saber cómo implementarlo según sus necesidades. Para tratar de entender el concepto de inyección de dependencias es necesario tener en cuenta otros conceptos, entre los cuales, los más importantes están los patrones de diseño e Inversión de Control (IoC).

2.2. Conceptos Relacionados con la Inyección de Dependencias

2.2.1. Inversión de Control (IoC)

La inversión de control es una forma de programar en la que el flujo de ejecución de un programa se invierte respecto a los métodos de programación tradicionales, en los que la interacción se expresa de forma imperativa realizando llamadas a procedimientos o funciones. De esta forma, para poder implementar la IoC, necesitamos un agente externo, normalmente llamado contenedor, que se encargará de realizar las conexiones necesarias entre las clases dependientes. Este agente externo será el encargado de controlar el flujo de la aplicación. De esa forma, podremos realiza la implementación de un código que esté débilmente acoplada.

2.2.2. Patrones de Diseño

Cuando se desarrolla una aplicación software es frecuente encontrarse en la situación de tener que volver a resolver problemas similares a otros que ya hemos solucionado anteriormente, y debemos volver a hacerlo partiendo de cero una y otra y otra vez (incluso dentro del mismo proyecto).

Debido a ello y basándose en la programación orientada a objetos surgieron los patrones de diseño, donde cada uno de ellos define la solución para resolver un determinado problema, facilitando además la reutilización del código fuente. Es una solución general reusable que puede ser aplicada a problemas que ocurren comúnmente en el desarrollo de software, es la descripción o plantilla de como resolver un problema que puede ser usada en diferentes situaciones. Los patrones de diseño son soluciones probadas, expresivas y fáciles de mantener. Muchos developers esta familiarizados con los Patrones de diseño, así que podemos decir que es un tipo de estándar de desarrollo.

2.2.3. Principio de Inversión de Dependencia

La principal ventaja de aplicar este principio es la flexibilidad que nos aporta. Esta flexibilidad también nos aporta más escalabilidad ya que vamos a poder sustituir componentes sin que los clientes que los consumen se vean afectados ya que dependen de la abstracción y no de la implementación concreta. Otro punto importante del principio es que las los detalles dependen de las abstracciones y no al revés. Es fundamental que la abstracción se defina en base a las necesidades del cliente y no en las capacidades de la implementación, de lo contrarío la abstracción estaría bastante acoplada a la implementación y teniendo así menos flexibilidad.

Ejemplo Visual, ¿Soldamos directamente una lámpara al cableado eléctrico de un pared?



No, creamos la abstracción enchufe y así podemos enchufar cualquier aparato electrónico.

2.3. Inyección de Dependencias

La inyección de dependencias (DI, por sus siglas en inglés) es un patrón usado en el diseño orientado a objetos de una aplicación. Es parte de uno de los cinco principios de diseño de clases conocido como S.O.L.I.D.

Como todo patrón de diseño, DI tiene como finalidad solucionar un problema común que los programadores encuentran en la construcción de aplicaciones. Este es, mantener los componentes o capas de una aplicación lo más desacopladas posible. Busca que sea mucho más sencillo reemplazar la implementación de un componente por otro. Así, evitar un gran cambio o impacto en la aplicación que pudiera originar que deje de funcionar por completo.

Para cumplir con dicho objetivo, DI nos permite inyectar comportamientos a componentes haciendo que nuestras piezas de software sean independientes y se comuniquen únicamente a través de una interface. Esto extrae responsabilidades a un componente para delegarlas en otro, estableciendo un mecanismo a través del cual el nuevo componente puede ser cambiado en tiempo de ejecución. Para lograr esta tarea DI se basa en un patrón más genérico llamado Inversión de Control (Inversion of Control).

Es necesario realizar ciertas modificaciones en el código fuente. Por ejemplo: el uso de interfaces que expongan la firmas de los métodos y operaciones de los que es responsable un componente, la eliminación de la instanciación de objetos o la necesidad de un modo de configuración que indique qué clases se instanciarán en el caso de solicitarlo.

En la inyección de dependencias, cuyo nombre [se le atribuye a Martin Fowler](http://www.martinfowler.com/articles/injection.html#FormsOfDependencyInjection), podemos encontrar cuatro roles:

* Las interfaces que definen las operaciones de los servicios - no necesariamente tienen que ser interfaces, sino que puede ser una clase abstracta
* Los servicios, que implementan las interfaces anteriores, que se van inyectar
* Los clientes que usarán los servicios
* El inyector, que es el encargado de construir e inyectar los servicios en los clientes

Estos roles no son completamente exclusivos, es decir, un cliente puede ser un servicio que deba ser inyectado a otro cliente.

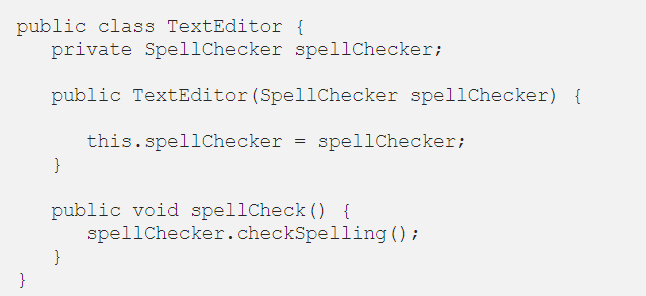
Adicionalmente también deberíamos saber que para implementar la inyección de dependencias (DI) hay muchos frameworks que realizan la tarea del inyector. Sin embargo, a estos frameworks usualmente se les conoce como Contenedores de DI, porque implementan una especie de contenedor o almacén en donde se guardan las dependencias para cuando necesiten ser resueltas en tiempo de ejecución.

Entre los frameworks, por ejemplo, para Java está Spring Boot, GWT y JSF, para JavaScript existe Angular, React.js y Vue.js.

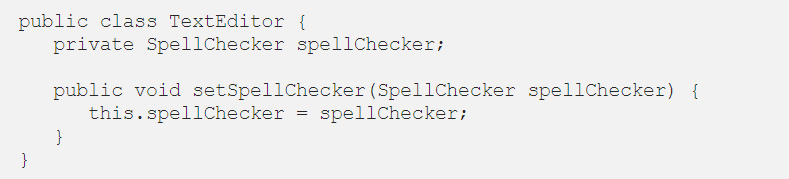
CASOS DESARROLLADOS

3.1. Formas de Inyectar Dependencias

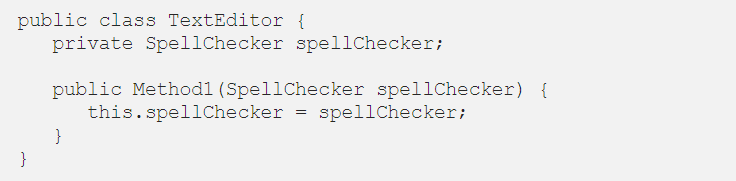
3.1.1. En el constructor



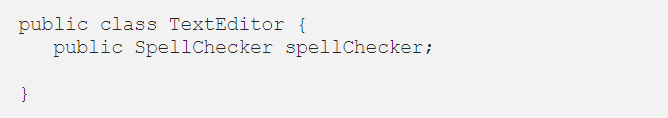
3.1.2. A través de un setter



3.1.3. A través de un método



3.1.4. A través de un campo



CONCLUSIONES

Con inyección de dependencias perdemos visión (no abstracta) en tiempo de diseño y con ello el detalle de lo que hace la aplicación. Además perdemos funcionalidades de depuración en módulos en proyectos externos ya que es preciso recompilar estos para poder modificarlos. Sin embargo, tu código se vuelve más modular y menos acoplado, ya que los submódulos no necesitan saber detalles de la implementación de otros en tu código. Producto de esta modularidad es que podemos hacer tests unitarios más fácilmente, cada parte del programa se puede aislar porque para realizar pruebas no tenemos que proveer a nuestros programas de las referencias reales, sino que podemos fingirlas. También podemos notar que existen diferentes maneras de poder implementar la inyección de dependencias, ya sea de forma manual o mediante el uso de framework, esto genera mayor eficiencia al momento de modificar una aplicación con el paso del tiempo por parte de los encargados en hacerlo, principalmente los desarrolladores.

RECOMENDACIONES

Comenzar a usarla puede ser tedioso, ya que necesitas escribir una abstracción de tus servicios además de su implementación, a veces podrías pensar que escribes código innecesariamente. Y no solo eso, porque luego de eso tienes que configurar tú el inyector. Sin mencionar que, si usas alguna librería, es muy probable que tengas que ejecutar tu aplicación para verificar que todas las dependencias se resuelvan correctamente en tiempo de ejecución, por tanto no es recomendable usarlo para aplicaciones que no son complejos

Bibliografía

Paradigmas de programación. (2020). Retrieved 19 February 2020, from https://ed.team/blog/paradigmas-de-programacion

REAL ACADEMIA ESPAÑOLA: Diccionario de la lengua española, 23.ª ed., [versión 23.3 en línea]. <https://dle.rae.es> [10 de febrero del 2020].

Paradigmas de programación. (2020). Retrieved 19 February 2020, from https://ed.team/blog/paradigmas-de-programacion

JuanDC. (2019). Aprende Inyección de Dependencias: El código es poder (Segunda parte). 2020, de Platzi Sitio web: https://platzi.com/blog/inyeccion-de-dependencias-el-codigo-es-poder/

Miguel Angel Alvarez. (2015). Inyección de dependencias. 2020, de DesarrolloWeb Sitio web: https://desarrolloweb.com/articulos/patron-diseno-contenedor-dependencias.html

Janneth Amaya. (2016). ¿Qué es y qué no es un Patrón de Diseño? 2020, de Medium Sitio web: https://blog.nearsoftjobs.com/qué-es-y-qué-no-es-un-patrón-de-diseño-487643d37a62