Recuerda marcar tu asistencia

Asistencia en LMS

Semana 1

Arquitectura de Software

Ing Edwin Garcia

El futuro digital es de todos

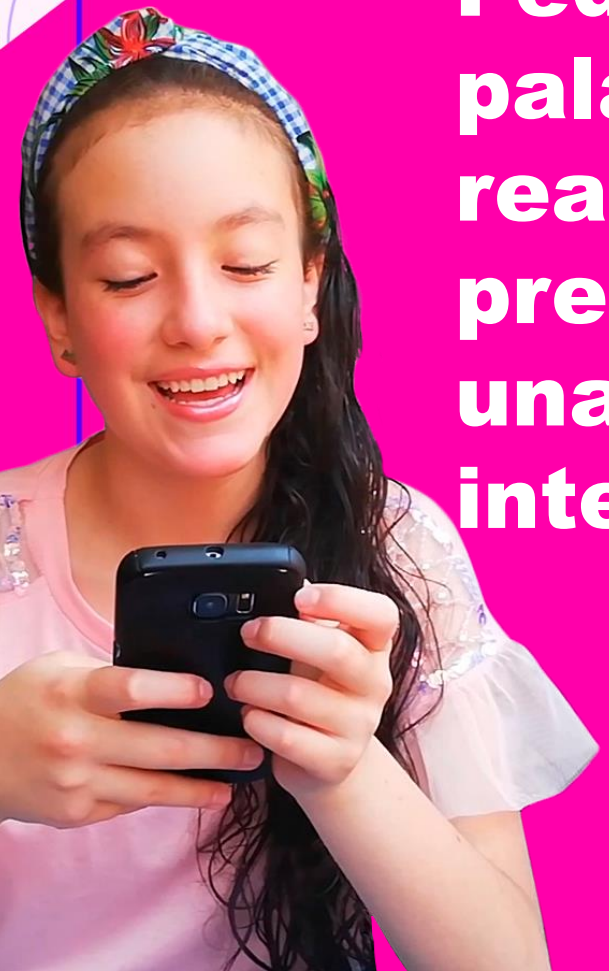
MinTIC

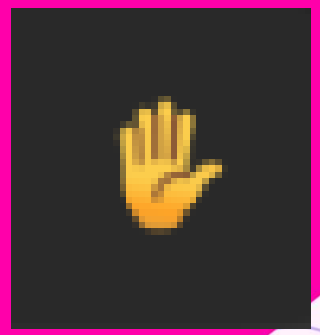
Hechos QUE CONECTAN

Recuerden marcar su asistencia

Asistencia en LMS

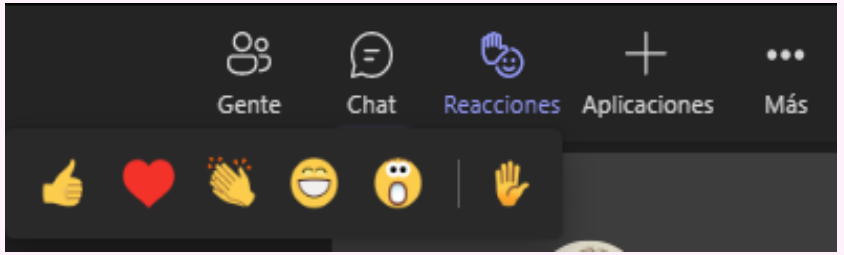
Pedir la palabra para realizar una pregunta o una intervención

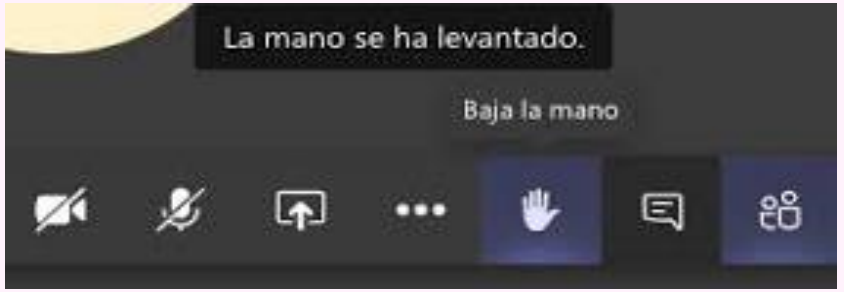




El futuro digital es de todos

MinTIC





Misión TIC 2022

unab
VIGILADA MINEDUCACIÓN

Hechos QUE CONECTAN

Debido a la cantidad de datos que se generan por segundo, es necesaria una serie de mecanismos que faciliten la administración de la información. Entre ellas se encuentra la Arquitectura de Software.

Sin embargo, este es un tema muy extenso, puesto que **la arquitectura abarca una gran variedad de herramientas, procesos, modelos y mecanismos** para llevar a cabo su tarea en la gestión de datos.

Por ello, llevar a cabo una arquitectura, a nivel general, requiere de una serie de conocimientos Big Data junto a una **preparación en guías y métodos de estructuración de la información.**

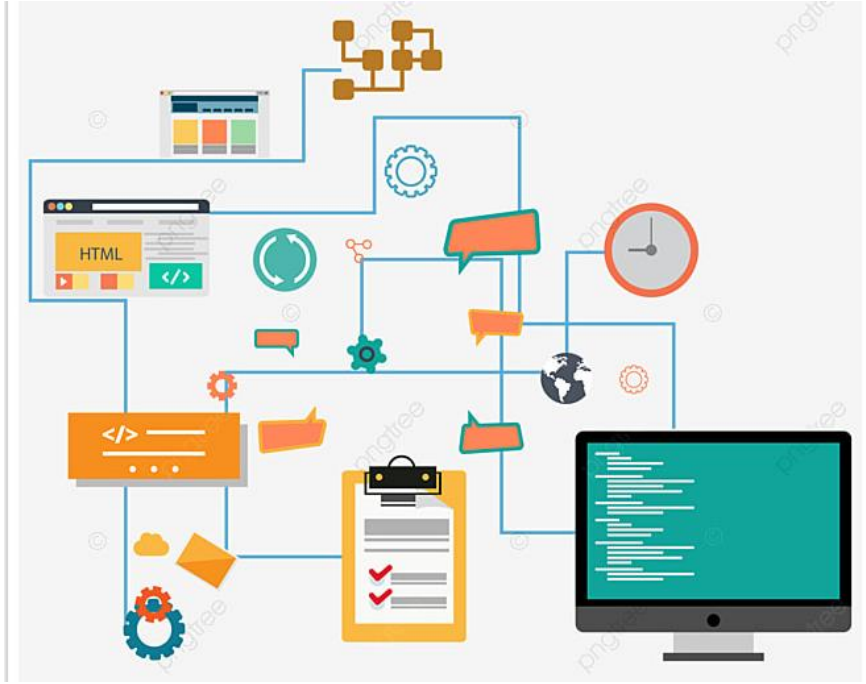
Arquitectura de Software

La arquitectura de Software hace referencia a la estructura y la relación entre las diferentes partes de un software y sus propiedades visibles externas.

En suma, una arquitectura de Software está compuesta por más arquitecturas de datos articuladas entre sí. Esta es la razón por la que abarca tantos elementos y herramientas para llevarse a cabo.

Su principal objetivo radica en ofrecer cierta calidad al sistema de administración de datos, a partir de su desempeño, ahorro de tiempo, su disponibilidad y usabilidad, la capacidad de modificarse y adecuarse a las nuevas necesidades del sistema, entre otros atributos de calidad.

Implantar una arquitectura de software capacitada para los intereses de una compañía deriva en la **reducción de costos, mejora en la toma de decisiones, efectividad de rutas de acción y proyección acertada.**



Por último, ten en cuenta que una **arquitectura de software** requiere de un determinado ciclo de etapas que constituyen sistemas: **requisitos, diseño, documentación y evaluación de su funcionamiento.**

Algunos términos de Arquitectura

Disponibilidad: se refiere a la posibilidad de acceder a la base de datos. Esta debería estar disponible todo el tiempo.

Latencia: se refiere a la distancia que puede existir entre los datos. La arquitectura de datos debe apuntar a una unificación.

Topología/conectividad/geografía: de igual manera, consta de la relación unificada de los datos entre sí.

Redundancia: un arquitecto de software necesita saber duplicar los servidores, porque si uno se cae tendrá que tener otro.

Gobierno: es el factor que administra la segmentación. Se debe tener en cuenta su influencia en el desarrollo de una arquitectura de datos.

Seguridad: la arquitectura de datos debe proteger la información en su desarrollo.

Elasticidad: un arquitecto de software debe poder gestionar desde poca cantidad de datos hasta un gran volumen.

Definición de la estructura

Requerimientos:

En esta etapa se recolecta la información y se documentan los requerimientos que influyen en la arquitectura de la aplicación, por ejemplo un color del menú no afecta al arquitectura, pero si por ejemplo la forma de guardar la información.

Diseño:

Es la etapa más crucial, aquí se define el uso de tecnologías adecuadas para resolver el problema y no solo porque una tecnología está de moda. También se tienen en cuenta patrones como por ejemplo MVC (Modelo, Vista, Controlador) o arquitectura de microservicios.

Documentación:

Una vez se ha definido el diseño es necesario comunicarlo de manera eficiente y eficaz a todos los involucrados, es importante crear documentación que sirva como referencia a todos y sea el marco de trabajo para todos.

Evaluación:

Es importante luego de tener la documentación evaluar, esto se puede hacer incluso sin haber hecho una línea de código y ver con todos los involucrados si hay algo en el diseño que pueda no funcionar y reformarlo, esta evaluación se debería hacer posterior teniendo métricas por ejemplo del rendimiento de la aplicación y saber si un cambio mejora o no dicho rendimiento.

Antes de hacer diagramas, definir requerimientos o empezar a programar, debemos entender el problema que vamos a resolver.

Todo sistema empieza con una idea. Existe un problema que podemos resolver generando valor para los usuarios y obteniendo un beneficio. En este punto también es importante definir qué vamos considerar éxito, así podremos enfocar todas nuestras decisiones en alcanzarlo.

El siguiente paso es **definir los requerimientos**, los dividimos en 3 grupos:

Requerimientos de negocio: son las reglas y funcionalidades más generales que necesita tu sistema.

Requerimientos de usuario: se relacionan con cómo se desenvuelven los usuarios usando la aplicación. También nos encargamos de los atributos de calidad, es decir, qué elementos específicos del sistema nos importan por sobre otros.

Requerimientos funcionales: se alimentan de las 3 capas de requerimientos anteriores para definir qué hay que hacer para conseguir cada funcionalidad en particular. También tienen en cuenta las restricciones y elementos operativos.

Patrones de Diseño de Software

¿Qué son los patrones de diseño de software?

- Si eres programador o estás despegando en el desarrollo informático y la programación, tarde o temprano te toparás con el término «**patrones de diseño de software**«.

¿Qué son los patrones de diseño / design patterns?

- Los patrones de diseño o *design patterns*, son una solución general, reutilizable y aplicable a diferentes problemas de diseño de software. Se trata de **plantillas que identifican problemas en el sistema y proporcionan soluciones apropiadas a problemas generales** a los que se han enfrentado los desarrolladores durante un largo periodo de tiempo, a través de prueba y error.

Historia de los patrones de diseño

En 1994, cuatro autores Erich Gamma, Richard Helm, Ralph Johnson y John Vlissides, a los que llamaron ***Gang of Four (GoF)***, publicaron un libro titulado ***Design Patterns***, elementos de software orientado a objetos reutilizables.

Historia de los patrones de diseño

Con este trabajo se inició el concepto de patrón de diseño en el desarrollo de software y recoge **23 patrones de diseño comunes.** Cada uno de ellos define la solución para resolver un determinado problema, facilitando además la reutilización del código fuente.

¿Por qué usar patrones de diseño?

El gran crecimiento del sector de las tecnologías de la información ha hecho que las prácticas de desarrollo de software evolucionen. Antes se requería completar todo el software antes de realizar pruebas, lo que suponía encontrarse con problemas. Para ahorrar tiempo y evitar volver a la etapa de desarrollo una vez que este ha finalizado, se introdujo una práctica de prueba durante la fase de desarrollo.

¿Por qué usar patrones de diseño?

Esta práctica se usa para identificar condiciones de error y problemas en el código que pueden no ser evidentes en ese momento. En definitiva, los patrones de diseño te **ayudan a estar seguro de la validez de tu código, ya que son soluciones que funcionan y han sido probados por muchísimos desarrolladores** siendo menos propensos a errores.

Claves sobre los patrones de diseño

Los patrones de diseño son una solución general reutilizable y aplicable a diferentes problemas de diseño de software.

Ayudan a validar el código, al ser soluciones que han sido probadas por numerosos desarrolladores antes.

Patrones de creación

Dan mecanismos de creación de objetos, aumentan la flexibilidad y reutilización del código existente.

Patrones estructurales

Facilitan soluciones y estándares eficientes con respecto a las composiciones de clase y estructurad de objetos.

Patrones de comportamiento

Se encargan de la comunicación entre objetos de clase.

Tipos de patrones de diseño de sotware

- Los patrones de diseño más utilizados se clasifican en tres categorías principales, cada patrón de diseño individual conforma un total de 23 patrones de diseño. **Las cuatro categorías principales son**:
- **Patrones creacionales**
- **Patrones estructurales**
- **Patrones de comportamiento**

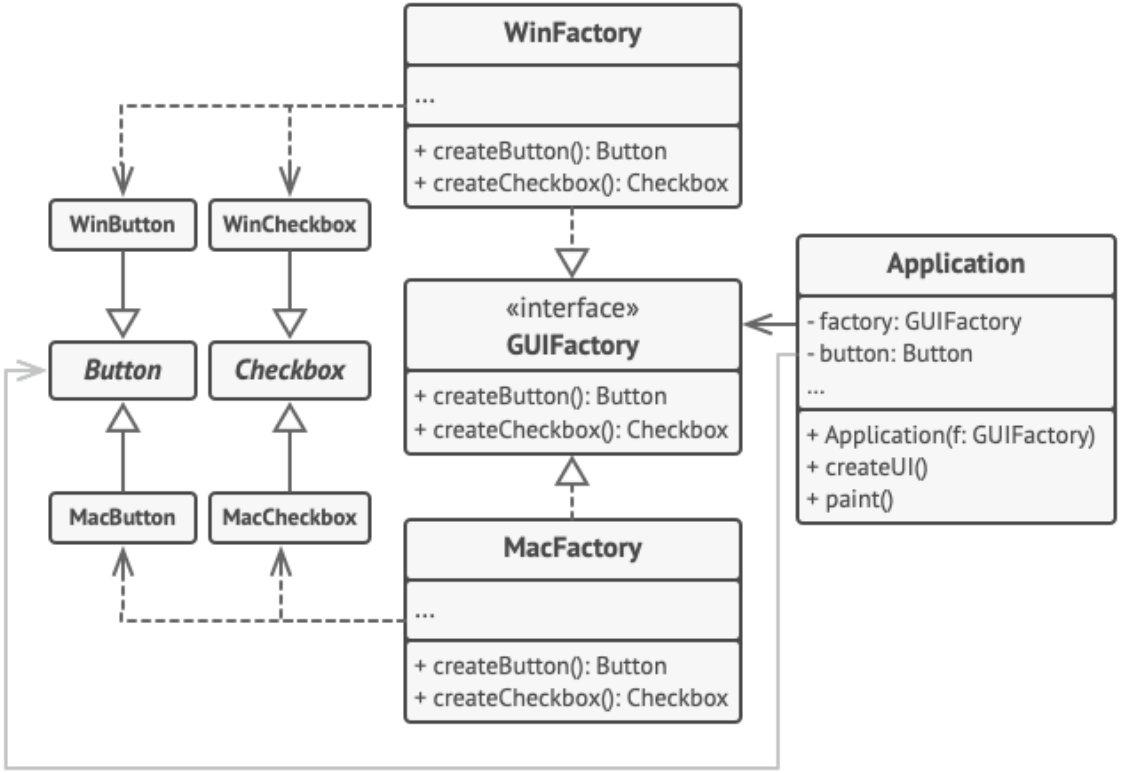
profile

Patrones creacionales

- Los patrones de creación proporcionan diversos mecanismos de creación de objetos, que aumentan la **flexibilidad y la reutilización del código existente de una manera adecuada a la situación**. Esto le da al programa más flexibilidad para decidir qué objetos deben crearse para un caso de uso dado.

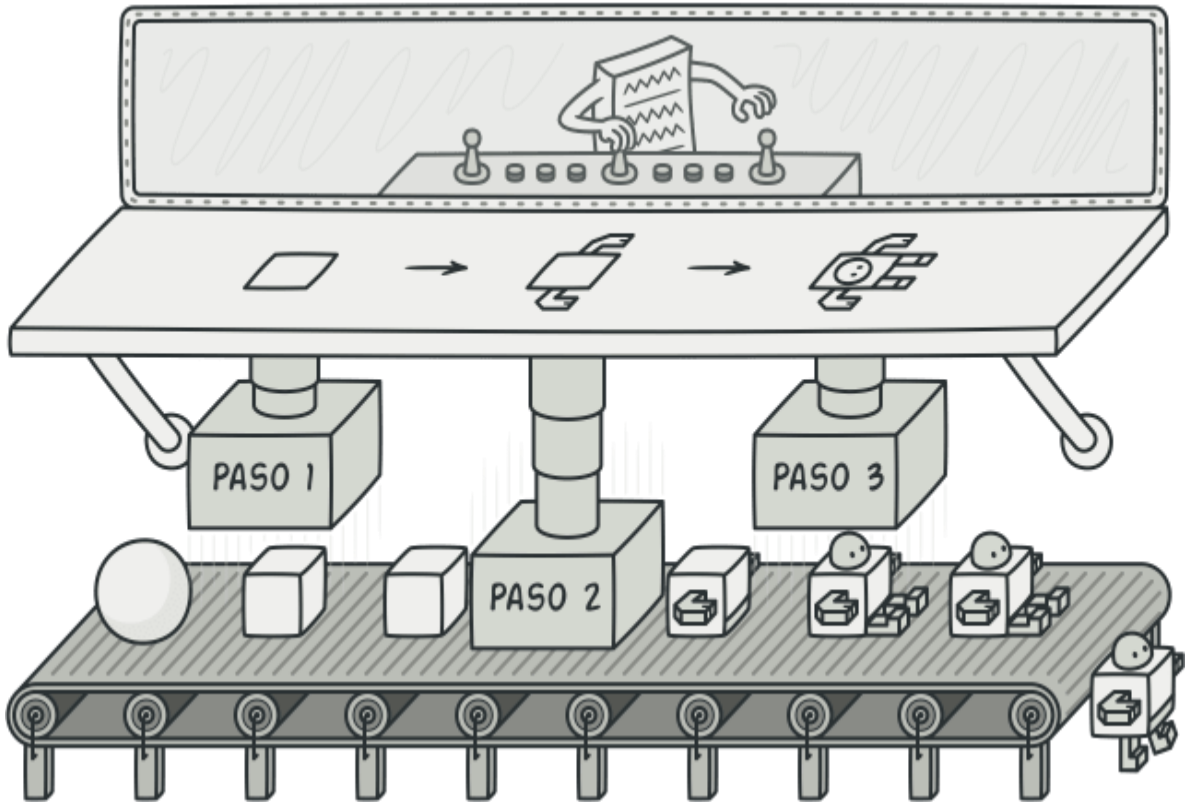
Abstract Factory

En este patrón, una interfaz crea conjuntos o familias de objetos relacionados sin especificar el nombre de la clase.



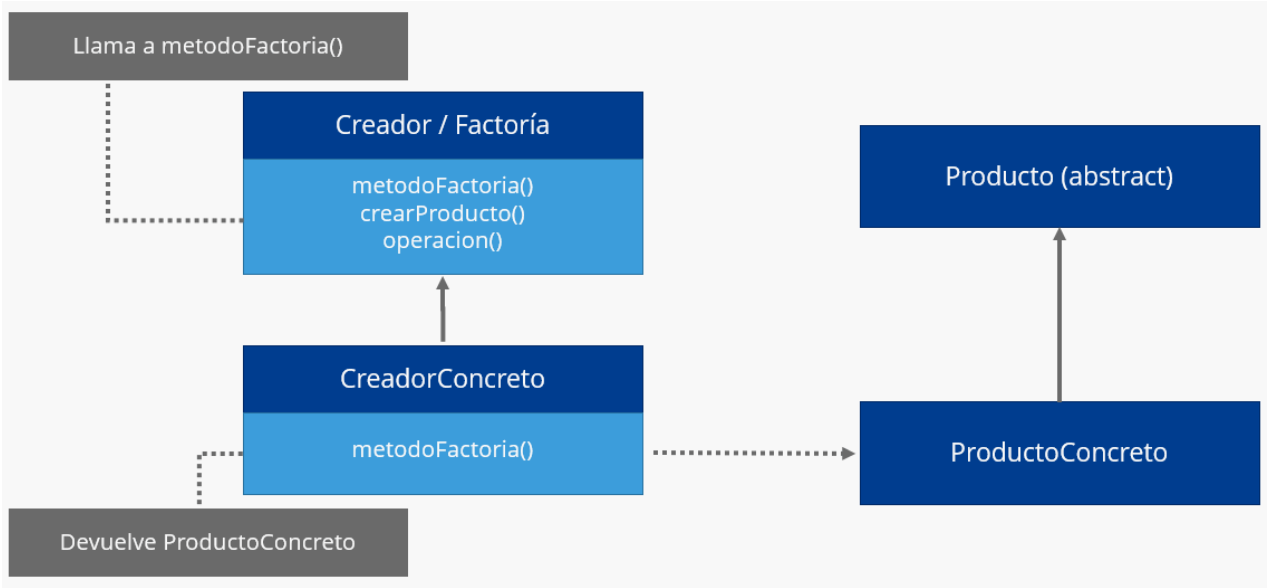
Builder Patterns

Permite producir diferentes tipos y representaciones de un objeto utilizando el mismo código de construcción. Se utiliza para la creación etapa por etapa de un objeto complejo combinando objetos simples. La creación final de objetos depende de las etapas del proceso creativo, pero es independiente de otros objetos.



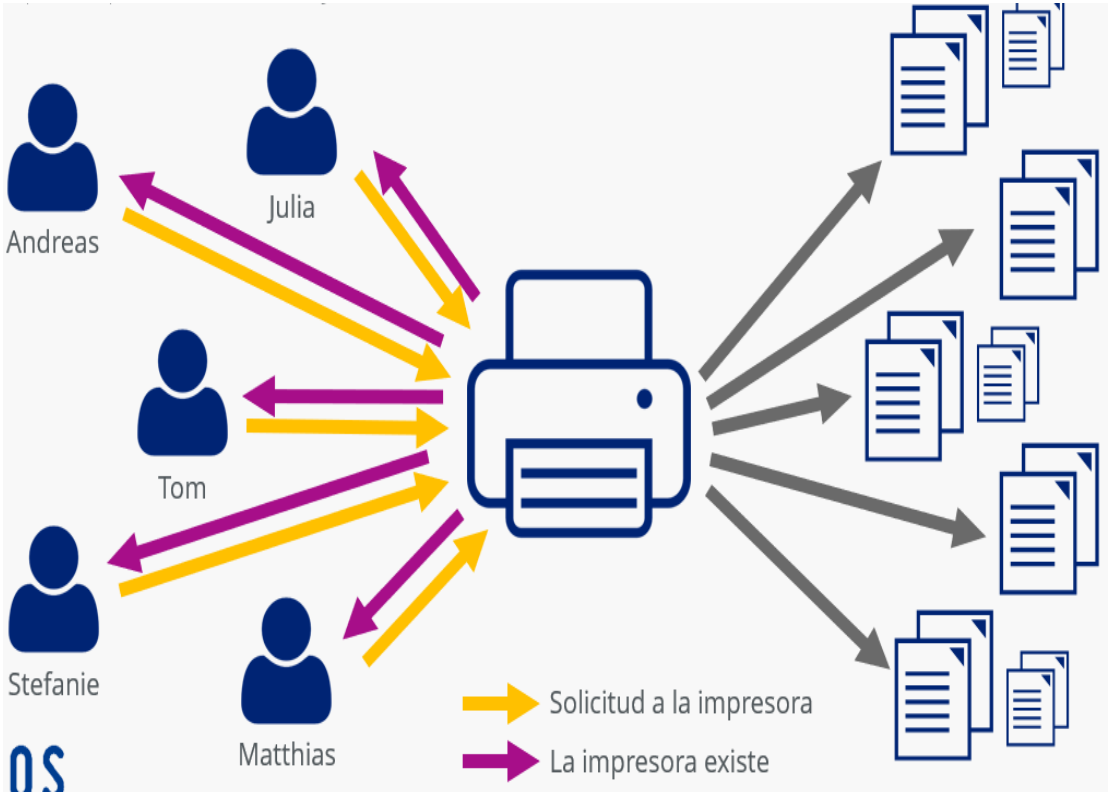
Factory Method

Proporciona una interfaz para crear objetos en una superclase, pero permite que las subclases alteren el tipo de objetos que se crearán. Proporciona instanciación de objetos implícita a través de interfaces comunes



Singleton

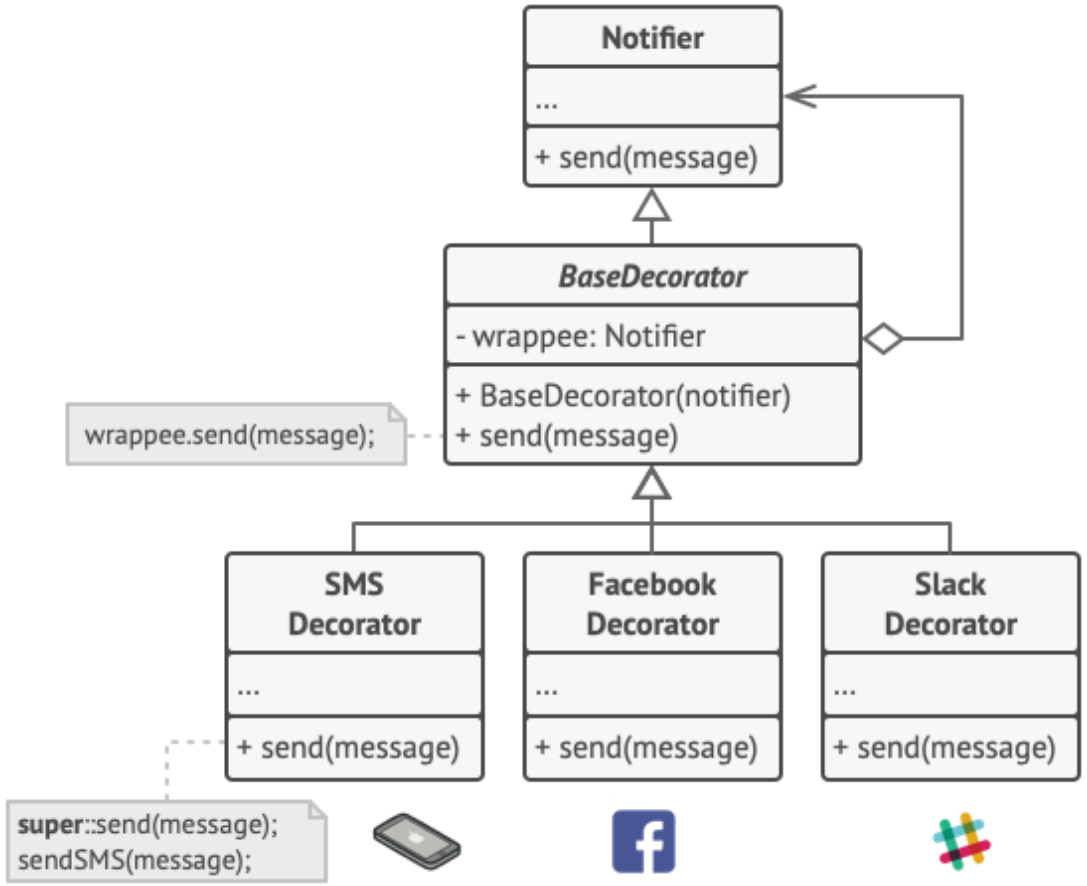
Este patrón de diseño restringe la creación de instancias de una clase a un único objeto.



Patrones estructurales

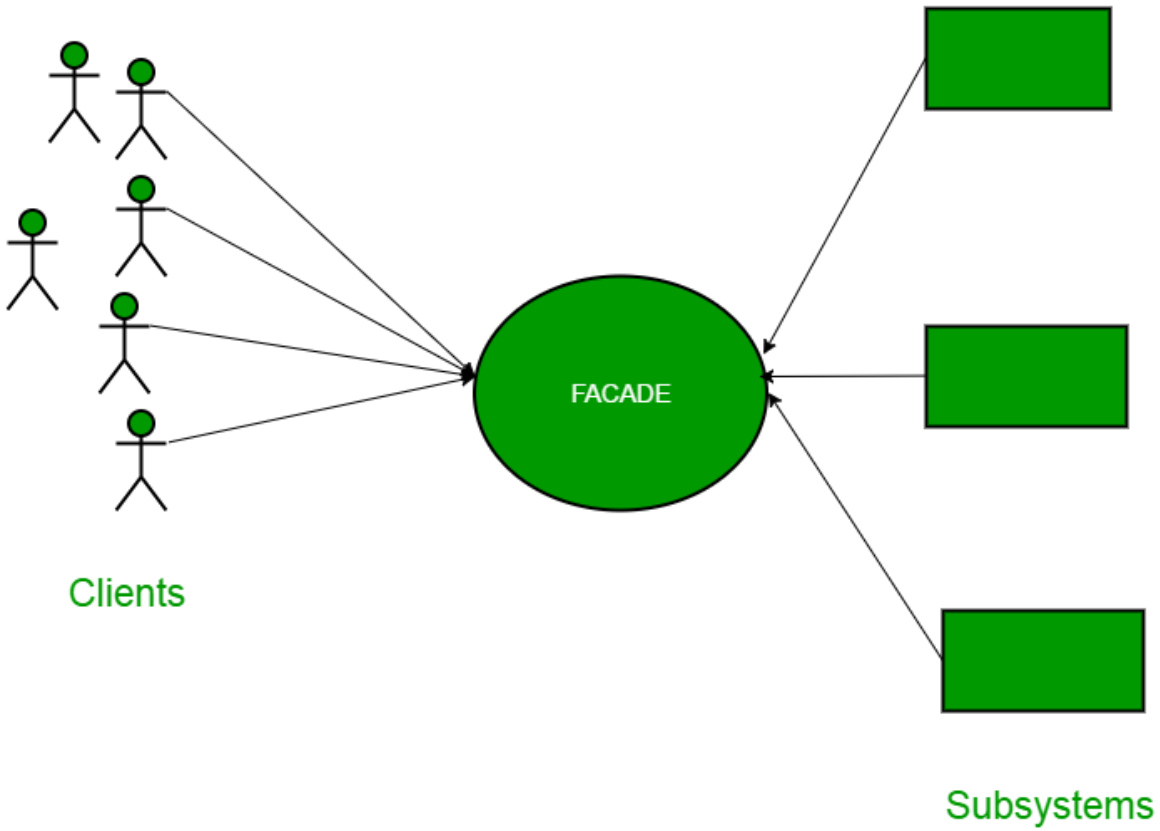
Decorator

Este patrón restringe la alteración de la estructura del objeto mientras se le agrega una nueva funcionalidad. La clase inicial permanece inalterada mientras que una clase *decorator* proporciona capacidades adicionales.



Facade

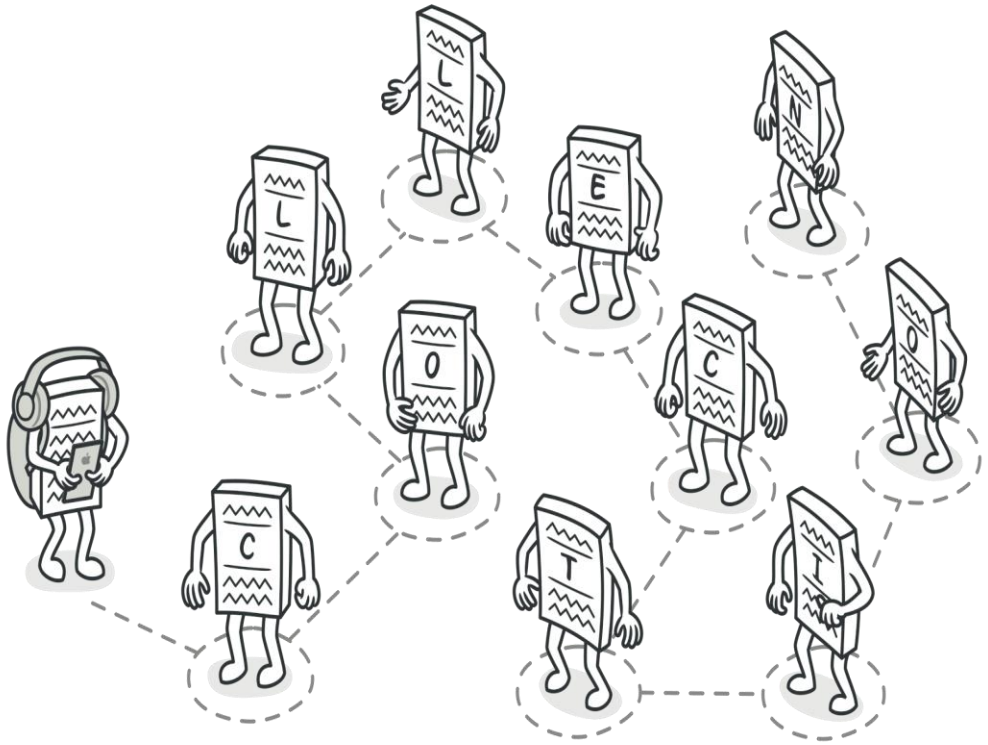
Proporciona una interfaz simplificada para una biblioteca, un marco o cualquier otro conjunto complejo de clases.



Patrones de comportamiento

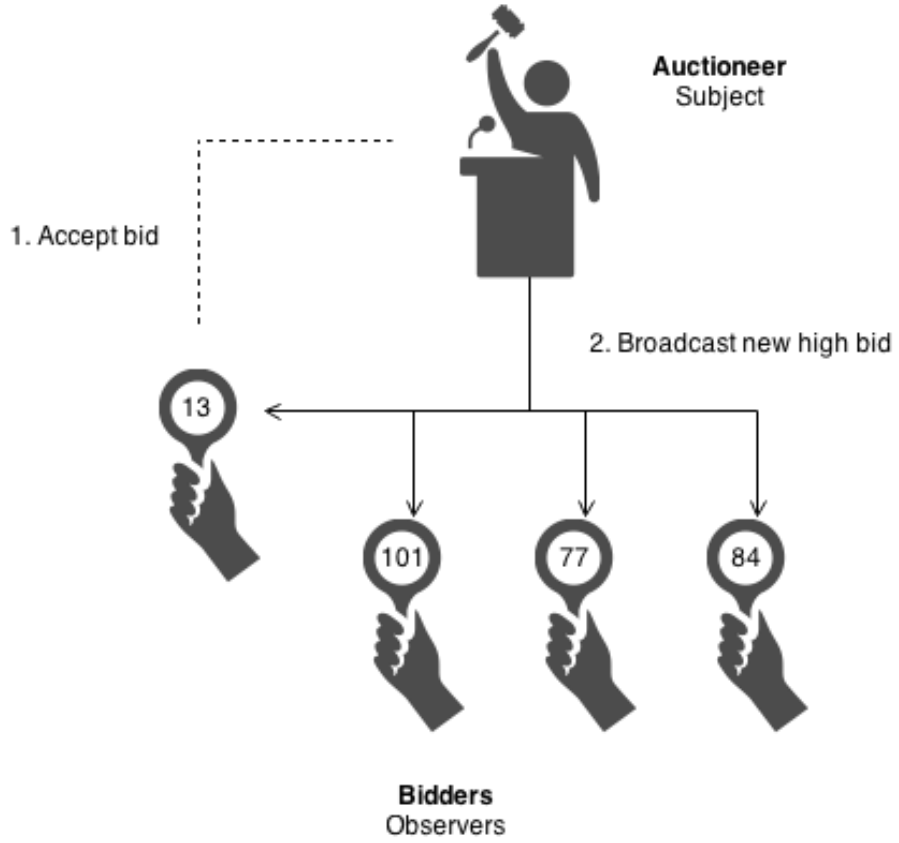
Iterator

Su utilidad es proporcionar acceso secuencial a un número de elementos presentes dentro de un objeto de colección sin realizar ningún intercambio de información relevante.



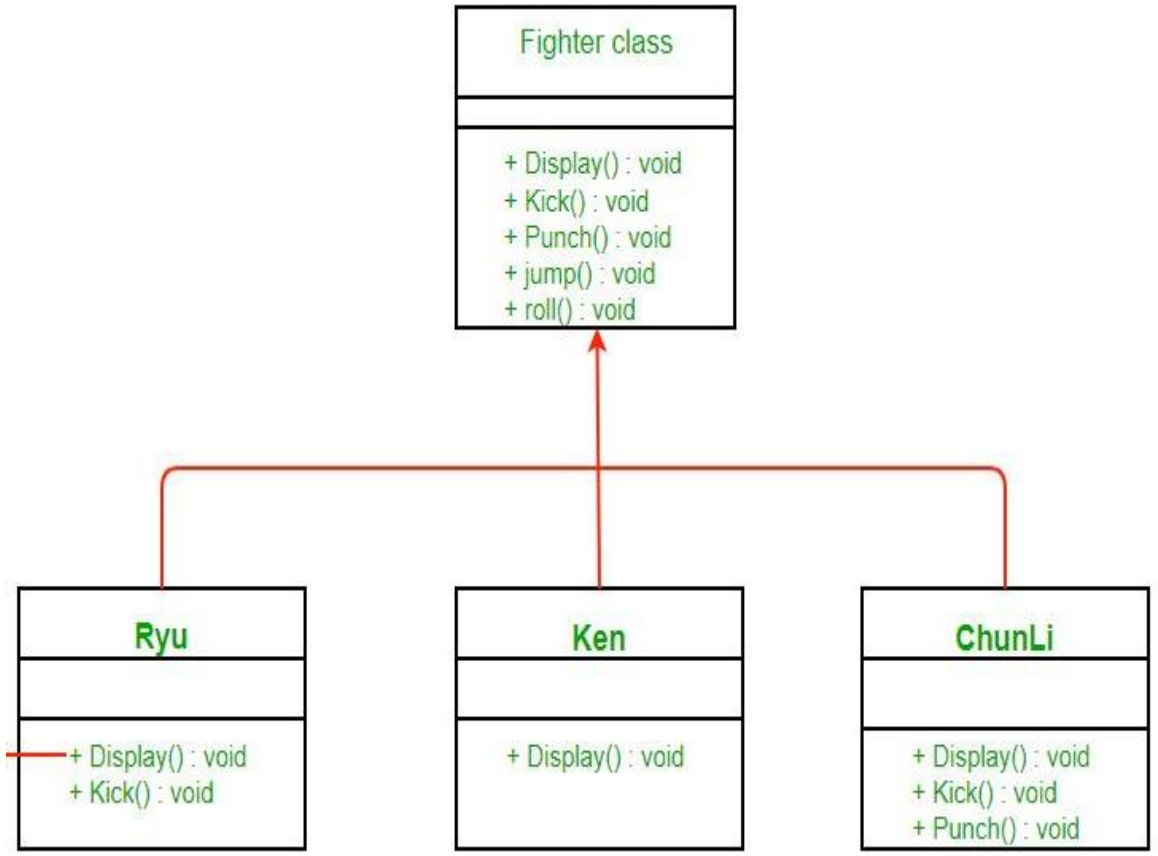
Observer

Permite definir un mecanismo de suscripción para notificar a varios objetos sobre cualquier evento que le suceda al objeto que está siendo observado.



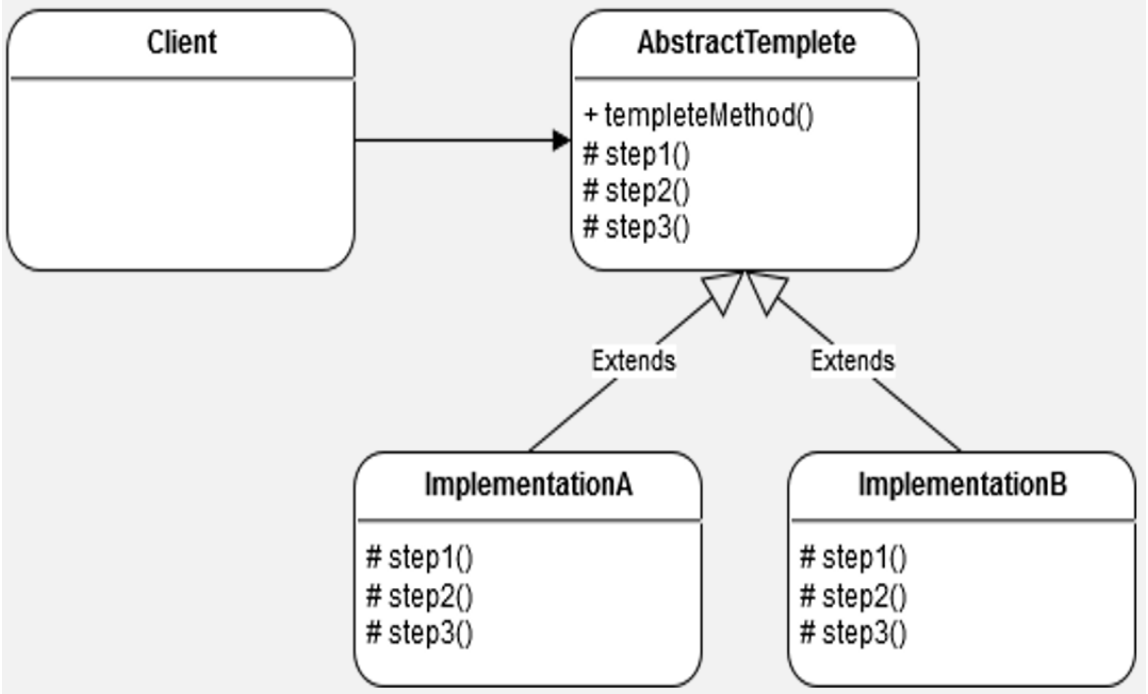
Strategy

Permite definir una familia de algoritmos, poner cada uno de ellos en una clase separada y hacer que sus objetos sean intercambiables.



Template method

Se usa con componentes que tienen similitud donde se puede implementar una plantilla del código para probar ambos componentes. El código se puede cambiar con pequeñas modificaciones.



Categoría GOF	Patrón de Diseño
Creacionales	1. Abstract Factory 2. Builder 3. Factory Method 4. Singleton
Estructurales	1. Decorator 2. Facade
De Comportamiento	1. Iterator 2. Observer 3. Strategy 4. Template Method

- https://keepcoding.io/blog/que-es-arquitectura-software/#Algunos_terminos_de_Arquitectura
- https://platzi.com/blog/que-es-arquitectura-de-software/
- https://www.youtube.com/watch?v=JI_THVXPToQ
- https://arquisoft.github.io/slides/course1819/ES.ASW.Te03_Documentacion.pdf
- https://profile.es/blog/patrones-de-diseno-de-software/
- https://www.youtube.com/watch?v=6_rl8z-OUVA

Bibliografía

El futuro digital es de todos

MinTIC

Practicar los temas tratados

MisionTIC2022

unab VIGILADA MINEDUCACIÓN

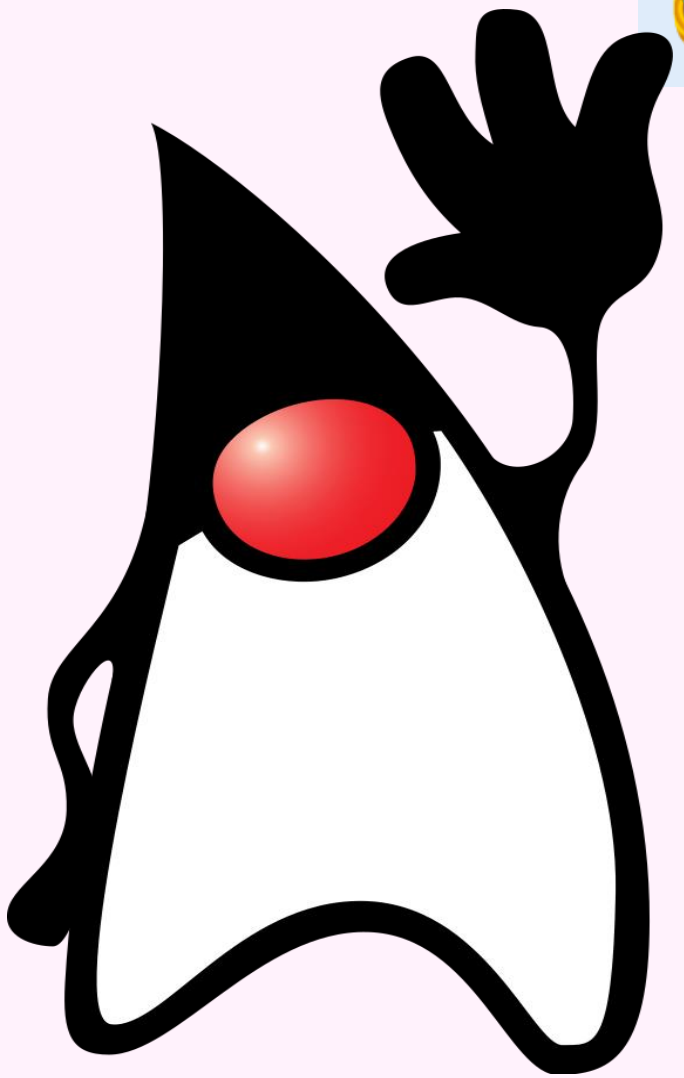
Hechos QUE CONECTAN

Nos vemos en la siguiente sesión

El futuro digital es de todos

MinTIC

Muchas Gracias



www.mintic.gov.co

MisionTIC2022

unab
VIGILADA MINEDUCACIÓN